cse250_project_Jacob_Padgett.md

Project 5 - Jacob Padgett

Elevator pitch

- There was a survey taken over 1000 people, And the surveyors want to be able to predict who has an annual income over $5C
  This project was made possible by Googles powerful search engine and YouTube. There was a request for graphs to be duplic
  but the data is the same.

TECHNICAL DETAILS

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

- In this section oh used a custom package that I created almost a year ago. It can be found here.
  - The purpose of this function is to remove any spaces from columns and substitute them with underscores. It also made a
    text lowercase. This makes it much easier to work with in Pandas.
- As we can see below, the columns have been altered from the original name to sure names and do you spaces have been substituted with other schools and everything is lowercase.

```
df.columns -->
['id', 'has_seen_any', 'fan', 'seen_episode_1', 'seen_episode_2','seen_episode_3', 'seen_episode_4',
'seen_episode_5',
'seen_episode_6','rank_episode_1', 'rank_episode_2', 'rank_episode_3', 'rank_episode_4','rank_episode_5',
'rank_episode_6',
'ranking_han_solo','ranking_luke_skywalker', 'ranking_princess_leia_organa','ranking_anakin_skywalker',
'ranking_obi_wan_kenobi',
'ranking_emperor_palpatine', 'ranking_darth_vader','ranking_lando_calrissian', 'ranking_boba_fett', 'ranking_c-
3p0','ranking_r2_d2',
 'ranking_jar_jar_binks', 'ranking_padme_amidala','ranking_yoda', 'shot_first',
'familiar_with_expanded_universe','fan_expanded_universe',
 'fan_star_trek', 'sex', 'age','household_income', 'education', 'location']
```

GRAND QUESTION 2 - Filter the dataset to those that have seen at least one film

- The original shape of the data was `(1188, 38)` (rows x columns).
- After filtering it to those who have seen at least one of the Star Wars films, I got `(936, 38)` As the shape of the data.
  - That means 252 people who filled out the survey did not see any of the films.

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

- Skip

GRAND QUESTION 4 - Clean and format the data so that it can be used in a machine learning model. Pl 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.

- What I did is used a dictionary and mapped the different values to have them be replaced with numbers.

- This little code gives me all the unique values in the `age` column, so I know I won't be missing any.
    - `df['age'].unique()`
- As can be seen below, the first one sets the range 18-29 to the number 1, and so forth.

```
age_range = {
    "18-29": 1,
    "30-44": 2,
    "45-60": 3,
    "> 60": 4
}
```

- As you may notice, the number zero is not in the above dictionary. However, I use the following code to substitute any values t were missing to the number zero.
    - `df['age_range'] = df['age_range'].fillna(0)`
- And as required, I dropped the original column with the following code.
    - `df.drop("age", axis = 1, inplace = True)`
- And here is a small sample of 25 with what that looked like (other columns are excluded for this view). We can see the first pe surveyed was between 30 - 44 years old and the last one was between 18 - 29 years old.

id	age_range
3290318577	2
3290845046	4
3289943097	3
3288522525	3
3288429092	4
3290681574	0
3288722710	3
3290176563	1
3289815685	2
3288482369	1
3290048884	3
3289993284	2
3290029531	4
3290779524	3
3289864687	3
3290364952	3
3289980534	4
3288458873	0
3290684868	2
3289582774	4

id	age_range
3289757385	2
3289693023	4
3290598307	3
3291684768	2
3292281095	1

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

- What I did is used a dictionary and mapped the different values to have them be replaced with numbers.
  - This little code gives me all the unique values in the `education` column, so I know I won't be missing any.
    - `df['education'].unique()`
  - As can be seen below, the first one (High school degree) sets the number 1, and so forth.

```
education_range = {
    "High school degree":1,
    "Some college or Associate degree":2,
    "Bachelor degree":3,
    "Graduate degree":4,
    "Less than high school degree":5,
}
```

- As you may notice, the number zero is not in the above dictionary. However, I use the following code to substitute any values t were missing to the number zero.
  - `df['education_range'] = df['education_range'].fillna(0)`
- And as required, I dropped the original column with the following code.
  - `df.drop("education", axis = 1, inplace = True)`
- And here is a small sample of 25 with what that looked like (other columns are excluded for this view). We can see the first pe surveyed marked their highest education as High school degree, and the last in this sample marked Some college or Associat degree.

id	education_range
3292879998	1
3292765271	1
3292763116	2
3292731220	2
3292719380	3
3292684787	1
3292663732	1
3292654043	2
3292640424	2
3292637870	0
3292609214	3

id	education_range
3292596911	4
3292587240	1
3292583038	2
3292580516	3
3292572872	2
3292565282	1
3292562297	2
3292522349	2
3292521066	3
3292511801	3
3292483200	3
3292465491	0
3292420123	2
3292384040	2

C - Create an additional column that converts the income ranges to a number and drop the income range categorical column

- What I did is used a dictionary and mapped the different values to have them be replaced with numbers.
  - This little code gives me all the unique values in the `household_income` column, so I know I won't be missing any.
    - `df['household_income'].unique()`
  - As can be seen below, the first one ($0 - $24,999) sets the number 1, and so forth.

```
household_income_range = {
    "$0 - $24,999":1,
    "$25,000 - $49,999":2,
    "$50,000 - $99,999":3,
    "$100,000 - $149,999":4,
    "$150,000+":5,
}
```

- As you may notice, the number zero is not in the above dictionary. However, I use the following code to substitute any values t were missing to the number zero.
  - `df['household_income_range'] = df['household_income_range'].fillna(0)`
- And as required, I dropped the original column with the following code.
  - `df.drop("household_income", axis = 1, inplace = True)`
- And here is a small sample of 25 with what that looked like (other columns are excluded for this view). We can see the first pe surveyed marked their income as between $0 - $24,999, and the last in this sample had the same income range.

id	household_income_range
3292879998	0
3292765271	1

id	household_income_range
3292763116	4
3292731220	4
3292719380	2
3292684787	0
3292663732	0
3292654043	1
3292640424	2
3292637870	0
3292609214	2
3292596911	3
3292587240	0
3292583038	1
3292580516	3
3292572872	0
3292565282	0
3292562297	3
3292522349	0
3292521066	0
3292511801	3
3292483200	1
3292465491	0
3292420123	4
3292384040	0

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

- As the goal is to predict someones income level, I used the following code to create a target.
  - `target = df['household_income_range']`

E - One-hot encode all remaining categorical columns.

- I ended up factorizing the data as opposed to using `pd.get_dummies()` from Pandas because factorizing it kept the dimensio (number of columns) small which provides less variance.
- Here is an example of the first 5 rows:

id	has_seen_any	fan	seen_episode_1	seen_episode_2	seen_episode_3	seen_episode_4	seen_episo
0	0	0	0	0	0	0	
1	0	1	0	0	0	-1	

id	has_seen_any	fan	seen_episode_1	seen_episode_2	seen_episode_3	seen_episode_4	seen_episoc
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	0	0	

GRAND QUESTION 5 - Build a machine learning model that predicts whether a person makes more thar $50k

- Ultimately, came down to this block of code that gave me an accuracy of almost 34% in predicting whether or not someone w makes over $50,000.
    - Be more precise, here's what the accuracy score was:
        - `accuracy_score = 0.33807829181494664`

```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=24)

# Creating random forest object
rf = RandomForestClassifier(random_state=24)

# Fit with the training data
rf.fit(X_train, y_train)

# Using the features in the test set to make predictions
y_pred = rf.predict(X_test)

# Comparing predictions to actual values
print(f'accuracy_score = {accuracy_score(y_test, y_pred)}')
```

APPENDIX A (PYTHON SCRIPT)

```
# %% [markdown]
# # Project 5: The war with Star Wars (4 days)
#
#
# %% [markdown]
# ## Background
# Survey data is notoriously difficult to handle.
# Even when the data is recorded cleanly the options for ‘write in questions’, ‘choose from multiple answers’,
‘pick all that are right’, and ‘multiple choice questions’
# makes storing the data in a tidy format difficult.
#
# 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>.
#
# A company would like to use this data to figure out if they can predict an interviewing job candidate’s curren
income based on a few responses about Star Wars movies.
# %% [markdown]
# ## GQ 1 - Shorten the column names and clean them up for easier use with pandas.

# %%
# Install and load my own package/library https://test.pypi.org/project/My-Cool-Functions/
import sys
!{sys.executable} -m pip install -i https://test.pypi.org/simple/ My-Cool-Functions==0.0.5
from my_data.ds_utilities import replace_spaces_with_underscore_in_column_names_and_make_lowercase
```

```
# Other imports
import pandas as pd


# %%
# My list of renamed columns
cols = [
'id',
'has seen any',
'fan',
'seen episode 1',
'seen episode 2',
'seen episode 3',
'seen episode 4',
'seen episode 5',
'seen episode 6',
'rank episode 1',
'rank episode 2',
'rank episode 3',
'rank episode 4',
'rank episode 5',
'rank episode 6',
'ranking Han Solo',
'ranking Luke Skywalker',
'ranking Princess Leia Organa',
'ranking Anakin Skywalker',
'ranking Obi Wan Kenobi',
'ranking Emperor Palpatine',
'ranking Darth Vader',
'ranking Lando Calrissian',
'ranking Boba Fett',
'ranking C-3P0',
'ranking R2 D2',
'ranking Jar Jar Binks',
'ranking Padme Amidala',
'ranking Yoda',
'shot first',
'familiar with Expanded Universe',
'fan Expanded Universe',
'fan Star Trek',
'Sex',
'Age',
'Household Income',
'Education',
'Location',
]


# %%
# Load, read data, change columns
url = 'https://github.com/fivethirtyeight/data/raw/master/star-wars-survey/StarWars.csv'
df = pd.read_csv(url, names=cols, encoding='unicode_escape')
df = replace_spaces_with_underscore_in_column_names_and_make_lowercase(df)

# Check the size of the data frame
df.shape


# %%
# Show the columns have been changed
df.columns

# %% [markdown]
# ## GQ 2 - Filter the dataset to those that have seen at least one film.

# %%
# Overwrite the data frame to those who have at least seen one film
df = df.query("has_seen_any == 'Yes'")
```

```
# Proof that there are fewer observations
df.shape


# %%
# Here we can see it worked
print(df.head(5).to_markdown(index=False))

# %% [markdown]
# ## GQ 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.

# %%


# %% [markdown]
# ## GQ 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 ma
in your report.
#
# - Create an additional column that converts the age ranges to a number and drop the age range categorical colu
# - Create an additional column that converts the school groupings to a number and drop the school categorical
column.
# - Create an additional column that converts the income ranges to a number and drop the income range categorica
column.
# - Create your target (also known as label) column based on the new income range column.
# - One-hot encode all remaining categorical columns.
# %% [markdown]
# ### A
# - Create an additional column that converts the age ranges to a number and drop the age range categorical colu

# %%
# See what the values are that need replaced
for i in df['age'].unique():
    print(i)


# %%
# Creating dictionary that maps rating to a number
age_range = {
    "18-29": 1,
    "30-44": 2,
    "45-60": 3,
    "> 60": 4
}

# Use map method to convert the strings to their corresponding numbers and save
df["age_range"] = df.age.map(age_range)

# NaNs to 0
df['age_range'] = df['age_range'].fillna(0)

# Drop the age
df.drop("age", axis = 1, inplace = True)

# View first 25
print(df[['id','age_range']].sample(25).to_markdown(index=False))

# %% [markdown]
# ### B
# - Create an additional column that converts the school groupings to a number and drop the school categorical
column.

# %%
# See what the values are that need replaced
for i in df['education'].unique():
    print(i)
```

```
# %%
# Creating dictionary that maps education to a number
education_range = {
    "High school degree":1,
    "Some college or Associate degree":2,
    "Bachelor degree":3,
    "Graduate degree":4,
    "Less than high school degree":5,
}

# Use map method to convert the strings to their corresponding numbers and save
df["education_range"] = df.education.map(education_range)

# NaNs to 0
df['education_range'] = df['education_range'].fillna(0)

# Drop the education
df.drop("education", axis = 1, inplace = True)

# View first 25
print(df[['id','education_range']].head(25).to_markdown(index=False))

# %% [markdown]
# ### C
# - Create an additional column that converts the income ranges to a number and drop the income range categorica
column.
#

# %%
# See what the values are that need replaced
for i in df['household_income'].unique():
    print(i)


# %%
# Creating dictionary that maps household_income to a number
household_income_range = {
    "$0 - $24,999":1,
    "$25,000 - $49,999":2,
    "$50,000 - $99,999":3,
    "$100,000 - $149,999":4,
    "$150,000+":5,
}

# Use map method to convert the strings to their corresponding numbers and save
df["household_income_range"] = df.household_income.map(household_income_range)

# NaNs to 0
df['household_income_range'] = df['household_income_range'].fillna(0)

# Drop the household_income
df.drop("household_income", axis = 1, inplace = True)

# View first 25
print(df[['id','household_income_range']].head(25).to_markdown(index=False))

# %% [markdown]
# ### D
# - Create your target (also known as label) column based on the new income range column.

# %%
target = df['household_income_range']

# %% [markdown]
# ### E
# - One-hot encode all remaining categorical columns.
# %%
# Check the types of each column and see what needs to be encoded still
df.dtypes

# %% [markdown]
# #### Get Dummies Wy

# %%
# # Change id column to numbers for ML reasons
# df['id'] = df['id'].astype(float)

# # Encode
# df = pd.get_dummies(df)

# # Reformat columns again
# df = replace_spaces_with_underscore_in_column_names_and_make_lowercase(df)

# # View first 25
# print(df.head(25).to_markdown(index=False))

# %% [markdown]
# #### Factorize Way

# %%
# Encode
df = df.apply(lambda x: pd.factorize(x)[0])

# Reformat columns again
df = replace_spaces_with_underscore_in_column_names_and_make_lowercase(df)

# View first 25
print(df.shape)
print(df.head(25).to_markdown(index=False))

# %% [markdown]
# ### Moving on

# %%
# Make everything the same dtype
df = df.astype(float)
df.dtypes

# %% [markdown]
# ## GQ 5 - Build a machine learning model that predicts whether a person makes more than $50k.

# %%
# Impoort the appropiate libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt


# %%
# # Sample 100
# df_sample = df.sample(10)
# target = df_sample['household_income_range']
# features = df_sample.drop('household_income_range', axis=1)
# sns.heatmap(df_sample);
# # sns.pairplot(df_sample);


# %%

# Drop id because of leakage
features = df.drop('id', axis=1)
```

```
# Establish the features
features = df.drop('household_income_range', axis=1)


# %%
# Splitting X and y variables into train and test sets using stratified sampling
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=24)

# Creating random forest object
rf = RandomForestClassifier(random_state=24)

# Fit with the training data
rf.fit(X_train, y_train)

# Using the features in the test set to make predictions
y_pred = rf.predict(X_test)

# Comparing predictions to actual values
print(f'accuracy_score = {accuracy_score(y_test, y_pred)}')


# %%
# features = df[['education_range',
# 'age_range',
# 'seen_episode_6_star_wars:_episode_vi_return_of_the_jedi',
# 'familiar_with_expanded_universe_no',
# 'fan_star_trek_no',
# 'sex_female']]


# %%
# Set up data in heatmap
sns.heatmap(df);


# %%
# Splitting X and y variables into train and test sets using stratified sampling
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=24)
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=24)

# Creating random forest object
rf = RandomForestClassifier(random_state=24)

# Fit with the training data
rf.fit(X_train, y_train)

# Using the features in the test set to make predictions
y_pred = rf.predict(X_test)

# Comparing predictions to actual values
print(f'accuracy_score = {accuracy_score(y_test, y_pred)}')


# %%
feat_imports = (pd.DataFrame(
    {"Feature Names": X_train.columns,
    "Importances": rf.feature_importances_})
    .sort_values("Importances", ascending=False))

print(feat_imports.to_markdown(index=False))


# %%

feat_imports.plot.bar(x='Feature Names', y='Importances', rot=90, width=.9,figsize=(20,10), title="Feature
Importance Ranking")
```

```
# %%
# Sample 100
df_sample = df.sample(10)
target = df_sample['household_income_range']
features = df_sample.drop('household_income_range', axis=1)
sns.heatmap(df_sample);
# sns.pairplot(df_sample);


# %%
for i in df.columns:
    print(i)


# %%
features = df[
              ['location',
               'age_range',
               'education_range',
               'household_income_range',
               'fan',
               'sex']]


# %%
import seaborn as sns
import matplotlib.pyplot as plt
# %matplotlib inline

# Correlation matrix
cor = df_sample.corr()
# cor = features.corr()

# PLotting Heatmap
# plt.figure(figsize = (10,6))
sns.heatmap(cor, annot=True)


# %%
sns.pairplot(features);

# %% [markdown]
# # Quiz Stuff

# %%
yes = df.query("has_seen_any == 'Yes'")
no = df.query("has_seen_any == 'No'")
print(yes.shape)
print(df.shape)
print(no.shape)
print(936+250)
print(936/(936+250))

print('-'*25)

yes = df.query("has_seen_any == 'Yes' & sex == 'Male'")
print(yes.shape)
no = df.query("has_seen_any == 'No' & sex == 'Male'")
print(no.shape)
print(423/(423+74))

# %%
```