Personality Powered AdBoost

transforming social media marketing with MBTI personality analytics

Project Background

- **Introversion (I) vs. Extroversion (E):** This dichotomy refers to how people direct and receive their energy. Extroverts tend to be outgoing, sociable, and energized by interacting with others, while introverts prefer to spend time reflecting, recharging, or engaging deeply with a few people.
- **Sensing (S) vs. Intuition (N):** This dichotomy refers to how people gather and process information. Sensors tend to rely on their five senses and concrete details, while Intuitives tend to rely on patterns, possibilities, and abstract concepts.
- **Thinking (T) vs. Feeling (F):** This dichotomy refers to how people make decisions. Thinkers tend to be logical and analytical, prioritizing objective criteria, while Feelers tend to consider personal values and emotions when making decisions.
- **Judging (J) vs. Perceiving (P):** This dichotomy refers to how people approach the outside world. Judgers tend to be decisive, organized, and prefer structure, while Perceivers tend to be flexible, adaptable, and prefer spontaneity.

Project Purpose

Dependencies and Data Source

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import seaborn as sns
%matplotlib inline
from matplotlib.pyplot import plot, figure, show, title
from sklearn.model_selection import StratifiedShuffleSplit,cross_
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score, make_scorer
from sklearn.ensemble import ExtraTreesClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.feature_extraction.stopwords import ENGLISH_STOP_WORDS
from sklearn.utils.extmath import safe_sparse_dot
from scipy import sparse, hstack
import os, sys, gc, warnings, pickle, time, gensim, logging, multiprocessing
warnings.filterwarnings("ignore")
%cd /content/drive/My Drive/Colab Notebooks/Personality Project
```

```
/content/drive/My Drive/Colab Notebooks/Personality Project
```

```
!head -n 3 mbti_1.csv
```

```
"type","posts"
"INFJ","http://www.personalitycafe.com/intuition-n-intuitive-forums/...
"ENTP","'I'm finding the lack of posts very alarming.|||Sex is neve...
"INTP","'Good luck|||Welcome and welcome to the board.|||Interesting...
```

Descriptive Analytics

```
data[:5].sample(5)
```

	type	posts
1	ENTP	'I'm finding the lack of posts very alarming.|||Sex is neve...
3	ENFJ	'Hahaha...|||...|||Yes|||Maybe|||Maybe|||Maybe|||Maybe...
0	INFJ	'In one of the posts I read, someone said that people generally type ...
2	INTP	'Good luck|||Welcome and welcome to the board.|||Interesting ...
4	ENTJ	'Yes|||Maybe|||Maybe|||Maybe|||Maybe|||Maybe|||Maybe|||Maybe...

```
# Introversion score and type
# Intuition score and type
# Thinking score and type
# Judging score and type
# Number of posts and type
# Number of words per post and type
# Number of unique words per post and type
# Number of links per post and type
# Number of image links per post and type
# Number of question marks per post and type
# Number of exclamation marks per post and type
# Number of ellipsis per post and type
# Number of capitalized words per post and type
# Number of capitalized letters per post and type
# Number of lowercase letters per post and type
# Number of punctuations per post and type
# Number of uppercase letters per post and type
# Number of stopwords per post and type
# Number of words per post and type
# Number of unique words per post and type
# Number of links per post and type
# Number of image links per post and type
# Number of question marks per post and type
# Number of exclamation marks per post and type
# Number of ellipsis per post and type
# Number of capitalized words per post and type
# Number of capitalized letters per post and type
# Number of lowercase letters per post and type
# Number of punctuations per post and type
# Number of uppercase letters per post and type
# Number of stopwords per post and type
```

```
plot_frequency_by_type('type', 'posts')
plot_percentage_by_type('type', 'posts')
```

```
# Introversion score and type
# Intuition score and type
# Thinking score and type
# Judging score and type
# Number of posts and type
# Number of words per post and type
# Number of unique words per post and type
# Number of links per post and type
# Number of image links per post and type
# Number of question marks per post and type
# Number of exclamation marks per post and type
# Number of ellipsis per post and type
# Number of capitalized words per post and type
# Number of capitalized letters per post and type
# Number of lowercase letters per post and type
# Number of punctuations per post and type
# Number of uppercase letters per post and type
# Number of stopwords per post and type
```

```
# Intuition score and type
# Thinking score and type
# Judging score and type
# Number of posts and type
# Number of words per post and type
# Number of unique words per post and type
# Number of links per post and type
# Number of image links per post and type
# Number of question marks per post and type
# Number of exclamation marks per post and type
# Number of ellipsis per post and type
# Number of capitalized words per post and type
# Number of capitalized letters per post and type
# Number of lowercase letters per post and type
# Number of punctuations per post and type
# Number of uppercase letters per post and type
# Number of stopwords per post and type
```

Word Cloud

```
def plot_word_cloud_per_group(df):
    for g in sorted(df.type.unique):
        print("Word Cloud for %s" %g)
        # Join all posts for this group
        text = ' '.join(df[df['type'] == g]['posts'])
        # Generate word cloud
        wordcloud = WordCloud(stopwords=ENGLISH_STOP_WORDS,
                              background_color='white',
                              max_words=2000,
                              max_font_size=40,
                              random_state=42,
                              collocations=False,
                              width=800,
                              height=400).generate(text)
        # Display the word cloud
        plt.figure(figure=(10, 5))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.show()
```

```
plot_word_cloud_per_group(df)
```

```
Word Cloud for ENFJ
Word Cloud for ENFP
Word Cloud for ENTJ
Word Cloud for ENTP
Word Cloud for ESFJ
Word Cloud for ESFP
Word Cloud for ESTJ
Word Cloud for ESTP
Word Cloud for INFJ
Word Cloud for INFP
Word Cloud for INTJ
Word Cloud for INTP
Word Cloud for ISFJ
Word Cloud for ISFP
Word Cloud for ISTJ
Word Cloud for ISTP
```

```
# Word Cloud for ENFJ
# Word Cloud for ENFP
# Word Cloud for ENTJ
# Word Cloud for ENTP
# Word Cloud for ESFJ
# Word Cloud for ESFP
# Word Cloud for ESTJ
# Word Cloud for ESTP
# Word Cloud for INFJ
# Word Cloud for INFP
# Word Cloud for INTJ
# Word Cloud for INTP
# Word Cloud for ISFJ
# Word Cloud for ISFP
# Word Cloud for ISTJ
# Word Cloud for ISTP
```

EJ

```
# Word Cloud for EJ
```

```
# Word Cloud for EJ
```

Introvert

music
family
friends
love
books
living
dream
small
song
games
human
listening
death
science
beautiful
nature
yeah
lol
thanks
haha
guys
us
crazy
laughing
oh
fun
xd
awesome
questions
thoughts
brain
heart
women
mind
ideas
hey
cool
sleep
eat
reality
art
life
time
work
good
people
world
love
friend
life
friends
sometimes
always
kind
thank
feeling
feel
music
family
thank
night
happy
nice
haha
lol
awesome
cute
cry
sad
hug
hugs
smile
laugh
tired
sleep
eat
food
cat
dog
dogs
cats
pet
pets
animal
animals
cute
love
life
friends
people
world
time
work
good
dream
reality
art
nature
beautiful
science
death
listening
human
games
books
living
small
song
family
friends
music
love

```
INFP      0.74      0.29      0.42       125
INFJ      0.71      0.20      0.31       177
ENTP      0.72      0.28      0.40       147
ENFP      0.00      0.00      0.00         9
ENTJ      0.00      0.00      0.00        18
ENFJ      0.00      0.00      0.00        12
ISTP      0.00      0.00      0.00        18
ISFP      0.41      0.65      0.50       206
INTP      0.41      0.54      0.59       268
INFP      0.54      0.73      0.65       461
INFJ      0.50      0.00      0.00        33
ENTP      0.50      0.02      0.04        42
ENFP      1.00      0.02      0.05        41
ENTJ      0.83      0.22      0.35        67

accuracy                           0.54      3725
macro avg      0.35      0.23      0.23      3725
weighted avg   0.54      0.54      0.48      3725
```

Classification Models - XGBoost

In [ ]: ~~time~~

```
model_xgb=XGBClassifier(max_depth=7,n_estimators=50,learning_rate=0.1)
model_xgb.fit(train_post,train_target)
```

```
CPU times: user 1min 2s, sys: 17.4 s, total: 1min 20s
Wall time: 2min 25s
```

```
*                                   XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=7, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=-1, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [ ]: ~~time~~

```
print('train classification report \n',classification_report(train_target,model_xgb.predict(train_post),target_names=target_encoder.inverse_transform)) for i in range(10)
print('test classification report \n',classification_report(test_target,model_xgb.predict(test_post),target_names=target_encoder.inverse_transform)) for i in range(10)
```

```
train classification report
              precision    recall  f1-score   support

      ENFJ       0.99      0.95      0.97       152
      ENFP       0.95      0.92      0.93       545
      ENTJ       0.99      0.91      0.95       185
      ENTP       0.99      0.93      0.96       147
      ISFJ       0.94      0.88      0.96        33
      ISFP       1.00      0.95      0.97        38
      INFJ       1.00      0.97      0.99       177
      INFP       1.00      0.97      0.99         5
      ISTP       0.92      0.91      0.92      1176
      INTP       0.94      0.93      0.94      1268
      ESFJ       0.92      0.93      0.92       873
      ESFP       0.91      0.92      0.91      1043
      ESTJ       0.99      0.92      0.95       239
      ESTP       0.99      0.92      0.95       217
      INTJ       0.99      0.93      0.96       164
      ISTJ       0.99      0.97      0.97       270

    accuracy                           0.93      6960
   macro avg       0.97      0.93      0.95      6960
weighted avg       0.93      0.93      0.93      6960

test classification report
              precision    recall  f1-score   support

      ENFJ       0.56      0.28      0.47        36
      ENFP       0.70      0.61      0.65       135
      ENTJ       0.67      0.40      0.50        45
      ENTP       0.58      0.56      0.56       127
      ISFJ       1.00      0.11      0.20         9
      ISFP       0.00      0.00      0.00         9
      INFJ       1.00      0.12      0.22         8
      INFP       0.50      0.28      0.36        18
      ISTP       0.63      0.63      0.63       294
      INTP       0.67      0.76      0.72       306
      ESFJ       0.67      0.47      0.47       219
      ESFP       0.63      0.63      0.63       261
      ESTJ       0.64      0.48      0.55        31
      ESTP       0.63      0.41      0.49        54
      INTJ       0.67      0.40      0.50        41
      ISTJ       0.65      0.41      0.51        67

    accuracy                           0.66      1725
   macro avg       0.65      0.46      0.55      1725
weighted avg       0.66      0.66      0.65      1725
```

Model Comparison

XGBoost performs better than random forest with higher accuracy, weighted avg precision and recall.
However, the classification model for 16 classes do not perform well due to unbalanced dataset.
The exact MBTI is also too specific for marketing purposes, it is more useful to just look at the four categories, and determine whether the customer is more extrovert or introvert? More thinking or feeling?

- Therefore we have built four classification model for each of the four MBTI categories

In [ ]: ~~time~~

Classification Models by Specific MBTI Categories

In [172]: data_X = data.copy()

```
data_X['First'] = (df1) for i in data_X.type]
data_X['Second'] = [i2] for i in data_X.type]
data_X['Third'] = [i3] for i in data_X.type]
data_X['Fourth'] = [i4] for i in data_X.type]
```

In [173]: data_X.head()

Out[173]:

	type	posts	first	second	third	fourth
0	INFJ	http://www.youtube.com/watch?v=qKNxw3knvQg...	I	N	F	J
1	ENTP	'I'm finding the lack of me in these posts very ...	E	N	T	P
2	INTP	'Good one _____ https://www.youtube.com/wat...	I	N	T	P
3	INFJ	'Dear INTJ I enjoyed our conversation the o...	I	N	T	J
4	ENTJ	'You're fired.|That's another silly misconce...	E	N	T	J

In [175]: train_data_text,test_data_text=train_test_split(data_X,test_size=0.2,random_state=42),stratify=data_X.type)

```
train_data_posts,train_length=clean_text(train_data)
test_data_posts,test_length=clean_text(test_data)

vectorizer=TfidfVectorizer(max_features=5000,stop_words='english',tokenizer=lambda x:x)
vectorizer.fit(train_data_posts)
```

```
100%|▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰| 3442/3442 [00:20<00:00, 2327.76it/s]
100%|▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰▰| 1136/1136 [00:00<00:00, 2351.89it/s]
```

Out[175]:

```
*                                   TfidfVectorizer
TfidfVectorizer(max_features=5000, stop_words='english',
                tokenizer=<__main__.Lemmatizer object at 0x7fab53efc310>)
```

In [176]: train_post=vectorizer.transform(train_data.posts).toarray()
test_post=vectorizer.transform(test_data.posts).toarray()

Classification for Extrovert (E) vs Introvert (I)

In [182]: target_encoder.fit(data[E0encoder])
train_target_e=target_encoder.fit_transform(train_data['First'])
test_target_e=target_encoder.fit_transform(test_data['First'])

In [183]: ~~time~~

```
model_xgb_E=XGBClassifier(n_jobs = -1, max_depth=5,n_estimators=50,learning_rate=0.1)
model_xgb_E.fit(train_post,train_target_e)
```

```
CPU times: user 1min 15s, sys: 1.4 s, total: 1min 20s
Wall time: 5.05 s
```

```
*                                   XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=-1, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [184]: print('train classification report \n',classification_report(train_target_e,model_xgb_E.predict(train_post),target_names=target_encoder.inverse_transform)) for i in range
print('test classification report \n',classification_report(test_target_e,model_xgb_E.predict(test_post),target_names=target_encoder.inverse_transform)) for i in range

```
train classification report
              precision    recall  f1-score   support

           E       0.95      0.85      0.90      1599
           I       0.91      0.99      0.95      5362

    accuracy                           0.92      6961
   macro avg       0.92      0.84      0.88      6961
weighted avg       0.92      0.92      0.92      6961

test classification report
              precision    recall  f1-score   support

           E       0.76      0.52      0.62       401
           I       0.87      0.95      0.91      1324

    accuracy                           0.85      1725
   macro avg       0.82      0.74      0.76      1725
weighted avg       0.84      0.85      0.84      1725
```

Classification for Sensing (S) vs Intuition (N)

In [185]: target_encoder.fit(data[S0encoder])
train_target_s=target_encoder.fit_transform(train_data['second'])
test_target_s=target_encoder.fit_transform(test_data['second'])

In [186]: ~~time~~

```
model_xgb_S=XGBClassifier(n_jobs = -1, max_depth=5,n_estimators=50,learning_rate=0.1)
model_xgb_S.fit(train_post,train_target_s)
```

```
CPU times: user 1min 16s, sys: 1.8 s, total: 1min 17s
Wall time: 10.1 s
```

```
*                                   XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=-1, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [187]: print('train classification report \n',classification_report(train_target_s,model_xgb_S.predict(train_post),target_names=target_encoder.inverse_transform)) for i in ran
print('test classification report \n',classification_report(test_target_s,model_xgb_S.predict(test_post),target_names=target_encoder.inverse_transform)) for i in ran

```
train classification report
              precision    recall  f1-score   support

           N       0.95      1.00      0.97      5983
           S       0.98      0.64      0.77       977

    accuracy                           0.95      6960
   macro avg       0.96      0.82      0.87      6960
weighted avg       0.95      0.95      0.94      6960

test classification report
              precision    recall  f1-score   support

           N       0.91      0.99      0.94      1495
           S       0.81      0.36      0.50       240

    accuracy                           0.90      1735
   macro avg       0.86      0.67      0.72      1735
weighted avg       0.89      0.90      0.88      1735
```

Classification for Thinking (T) vs Feeling (F)

In [188]: target_encoder.fit(data[T0encoder])
train_target_t=target_encoder.fit_transform(train_data['third'])
test_target_t=target_encoder.fit_transform(test_data['third'])

In [189]: ~~time~~

```
model_xgb_T=XGBClassifier(n_jobs = -1, max_depth=5,n_estimators=50,learning_rate=0.1)
model_xgb_T.fit(train_post,train_target_t)
```

```
CPU times: user 1min 16s, sys: 2.01 s, total: 1min 21s
Wall time: 11.6 s
```

```
*                                   XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=-1, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [190]: print('train classification report \n',classification_report(train_target_t,model_xgb_T.predict(train_post),target_names=target_encoder.inverse_transform)) for i in ran
print('test classification report \n',classification_report(test_target_t,model_xgb_T.predict(test_post),target_names=target_encoder.inverse_transform)) for i in range

```
train classification report
              precision    recall  f1-score   support

           F       0.92      0.92      0.92      3755
           T       0.90      0.91      0.91      3205

    accuracy                           0.91      6960
   macro avg       0.91      0.91      0.91      6960
weighted avg       0.91      0.91      0.91      6960

test classification report
              precision    recall  f1-score   support

           F       0.84      0.82      0.84       939
           T       0.80      0.82      0.81       786

    accuracy                           0.82      1725
   macro avg       0.82      0.82      0.82      1725
weighted avg       0.82      0.82      0.82      1725
```

Classification for Judging (J) vs Perceiving (P)

In [191]: target_encoder.fit(data[J0encoder])
train_target_j=target_encoder.fit_transform(train_data['fourth'])
test_target_j=target_encoder.fit_transform(test_data['fourth'])

In [192]: ~~time~~

```
model_xgb_J=XGBClassifier(n_jobs = -1, max_depth=5,n_estimators=50,learning_rate=0.1)
model_xgb_J.fit(train_post,train_target_j)
```

```
CPU times: user 1min 14s, sys: 1.48 s, total: 1min 21s
Wall time: 11.6 s
```

```
*                                   XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=-1, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [193]: print('train classification report \n',classification_report(train_target_j,model_xgb_J.predict(train_post),target_names=target_encoder.inverse_transform)) for i in ran
print('test classification report \n',classification_report(test_target_j,model_xgb_J.predict(test_post),target_names=target_encoder.inverse_transform)) for i in range

```
train classification report
              precision    recall  f1-score   support

           J       0.91      0.78      0.84      2747
           P       0.87      0.95      0.91      4213

    accuracy                           0.88      6960
   macro avg       0.89      0.87      0.87      6960
weighted avg       0.88      0.88      0.88      6960

test classification report
              precision    recall  f1-score   support

           J       0.81      0.71      0.76       687
           P       0.81      0.88      0.85      1038

    accuracy                           0.81      1725
   macro avg       0.81      0.79      0.79      1725
weighted avg       0.81      0.81      0.81      1725
```

Put every category classification in one dataframe

```
# Define the XGBoost models to evaluate
models = {
    'All 16-class Classifications': model_xgb, train_target, test_target,
    'Extrovert vs Introvert': model_xgb_E, train_target_e, test_target_e,
    'Sensing vs. Intuition': model_xgb_S, train_target_s, test_target_s,
    'Thinking vs. Feeling': model_xgb_T, train_target_t, test_target_t,
    'Judging vs. Perceiving': model_xgb_J, train_target_j, test_target_j
}

# Evaluate each model and store the results in a dataframe
results = pd.DataFrame(columns=['Model', 'Train Accuracy', 'Test Accuracy'])
for name, model, train, test in models:
    report1 = classification_report(train, model.predict(train_post), output_dict=True)
    report2 = classification_report(test, model.predict(test_post), output_dict=True)
    results = results.append({'Model': name, 'Train Accuracy': '{:.3%}'.format(report1['accuracy'])}, 'Test Accuracy': '{:.3%}'.format(report2['accuracy'])}, ignore_index=Tru

# Print the results dataframe
results
```

Out[196]:

	Model	Train Accuracy	Test Accuracy
0	All 16-class Classifications	92.97%	66.22%
1	Extrovert vs Introvert	92.10%	85.24%
2	Sensing vs. Intuition	94.83%	90.03%
3	Thinking vs. Feeling	91.26%	82.03%
4	Judging vs. Perceiving	88.28%	80.82%

Conclusion

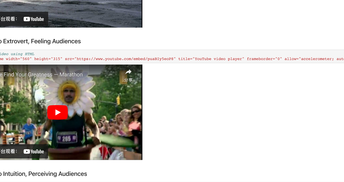
- This *Adboost Engine* analyzes the customer MBTI personality from their social media presences.
- Companies can use MBTI categories to design marketing campaigns for consumers based on different personality types.
- By integrating the Adboost Engine with a targeted advertising system, companies can display ads that are specifically tailored to the personality traits of their audience, increasing the likelihood that the ads will resonate and result in conversions.

Appeals to Introvert, Thinking Audiences

In [198]: from IPython.display import HTML

```
# Embed a video using HTML
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/i4K1k-STFqY" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard
```

Out[198]:

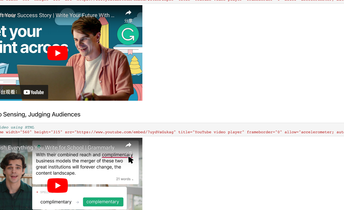


Appeals to Extrovert, Feeling Audiences

In [199]: ~~time~~

```
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/QuaAhlSwoO8" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard
```

Out[199]:



Appeals to Intuition, Perceiving Audiences

In [200]: ~~time~~

```
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/2iKzEP8qnQ8" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard
```

Out[200]:

Appeals to Sensing, Judging Audiences

In [201]: ~~time~~

```
# Embed a video using HTML
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/TvpPxaOwkqY" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard
```

Out[201]: