Planet Terp Prediction

Modou Sarr, CMSC 320, April 28 2025

Table of Contents

Overview

What is Planet Terp?

An interactive course-planning tool for UMD students

Integrates course catalogs, schedules, and student feedback

Why this problem is interesting:

Students often rely on anecdotal advice when choosing professors

High variability in teaching styles, grading, and workload

Predicting ratings from past reviews can help guide better course decisions

Our goal:

Automate rating prediction to surface “hidden gems.”

Enhance Planet Terp’s recommendations with data-driven insights

DATA

What Data Are We Using

Source:

Scraped RateMyProfessor data (JSON) for all UMD instructors
Via Planet Terp's pipeline (`professors.json`)

Scope:

13,427 unique professors,

37,553 student reviews

Key fields extracted:

`target_rating`–professor's overall average rating

`course`–course code

`review_text`–raw written feedback

`grade_letter` & **`grade_pt–reported/`**converted grade

`vader_compound`–sentiment polarity (VADER)

What Data Are We Using

By structuring the raw RateMyProfessor JSON into a review-level DataFrame, we capture both quantitative ratings and grades and qualitative text and sentiment signals for our modeling.

Top Variables

1. Grade Point
Students' earned/expected GPA in the class is the single strongest predictor—higher grades → higher ratings.

2. Review Sentiment
VADER "compound" score of the review text: the more positive the language, the higher the predicted rating.

3. Course-level introductory (100-level) vs. advanced (300-/400-level) courses show systematic rating differences.

4. Review Length
Longer, more detailed reviews carry extra signal about teaching effectiveness.

5. Subject Area
Departmental effects: some courses (e.g., ENAE vs. PHYS) tend to be rated higher or lower on average.

MODELS AND HOW THEY DID

Models

Models	MAE	R^2
KNN	0.348	0.800
Random Forest Regressor	0.333	0.810
Linear Regression	0.324	0.828

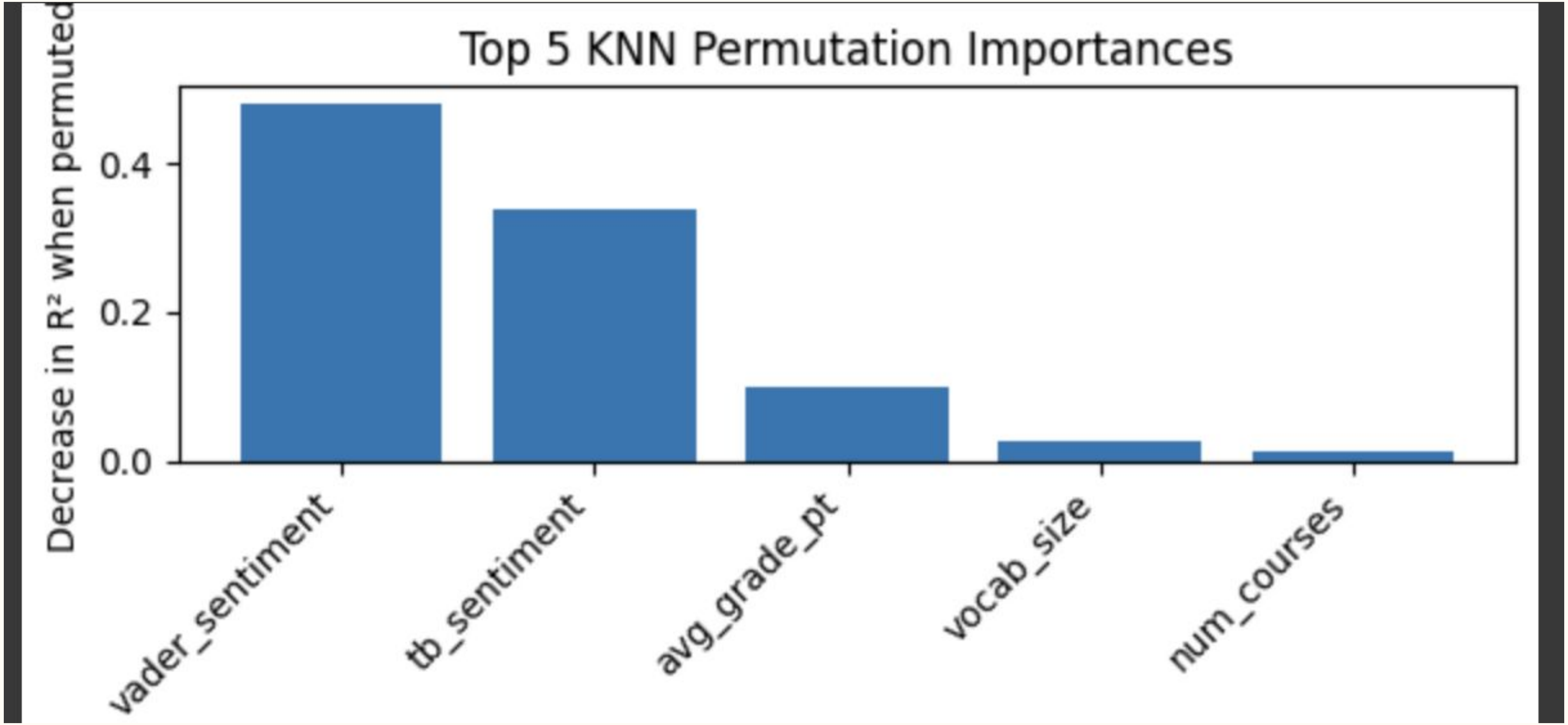
KNN Model

- **How far off are we?**
  On average, our KNN guesses are about 0.35 stars away from a professor's real rating.

- **How well does it get the patterns?**
  It captures roughly 80% of what makes students rate a professor the way they do.

- **Understanding:**
  If a professor actually has a 4.0 rating, KNN will usually predict somewhere between 3.65 and 4.35.

Permutation Importance



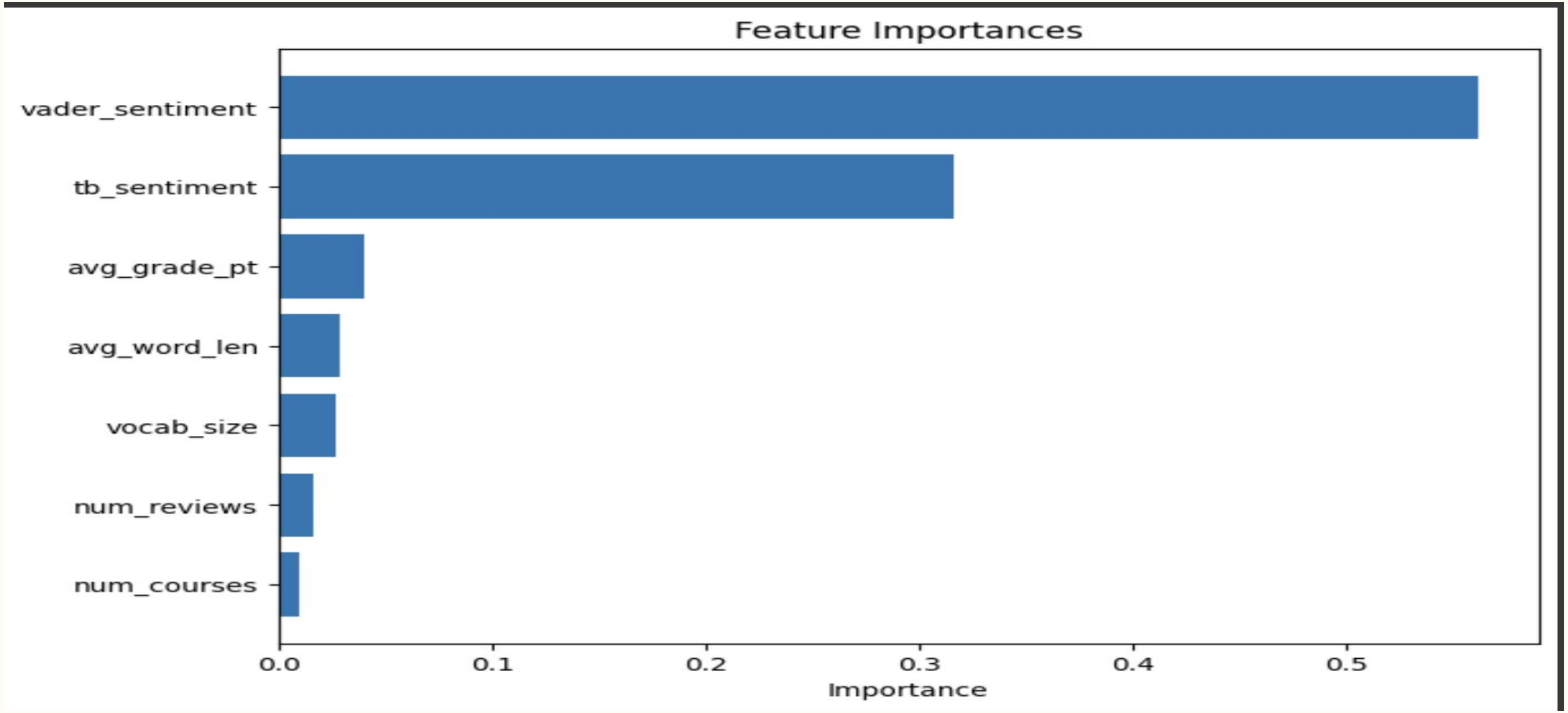
Random Forest Model

Average error: about 0.33 stars off.

Pattern captured: about 81% of the factors affecting ratings.

Understanding:
Random Forest is a bit smarter than KNN, it makes slightly smaller mistakes and understands a bit more of why ratings move up or down.

Feature Importance



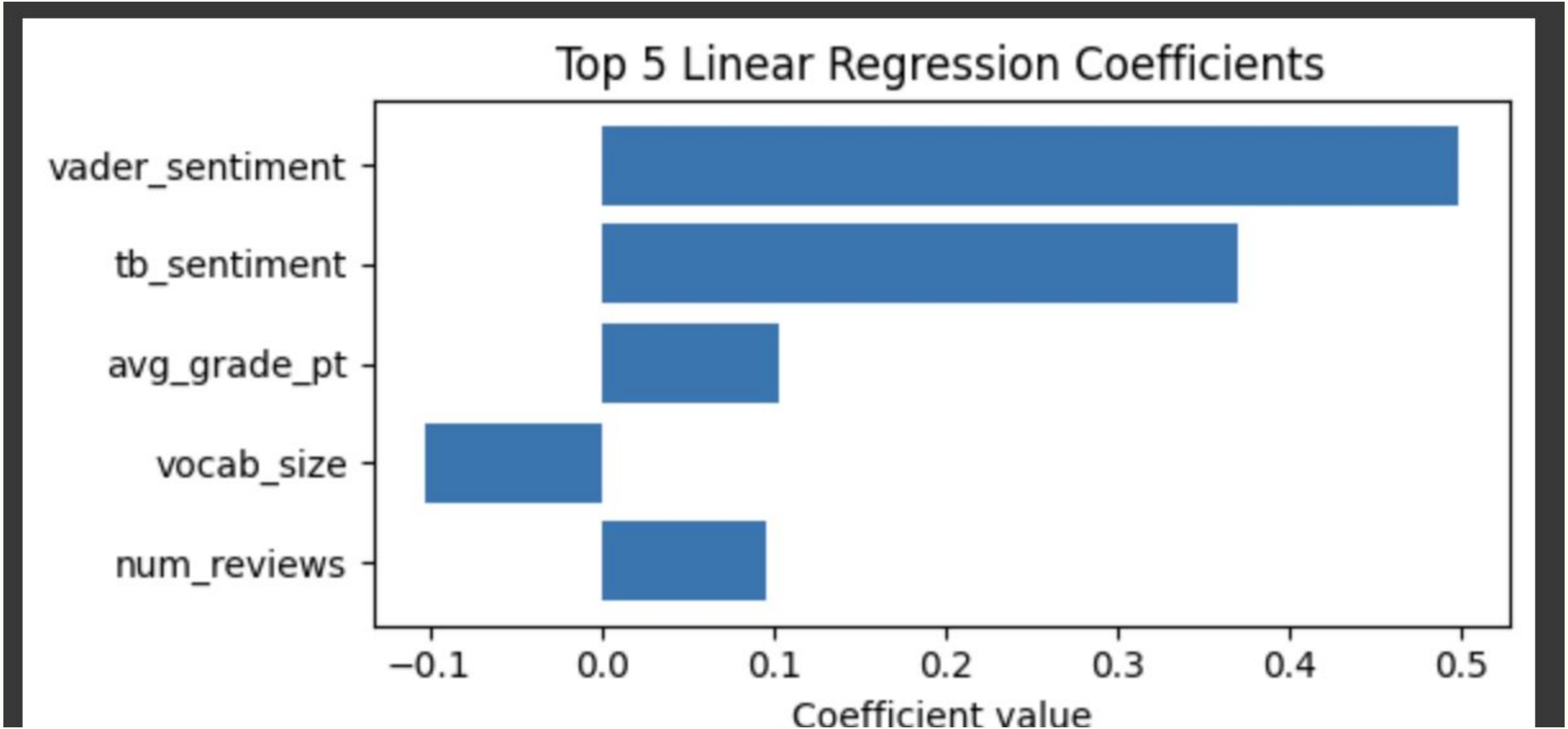
Linear Regression

Average error: about 0.32 stars off.

Pattern captured: about 83% of the reasons behind students’ ratings.

Understanding:
Even a simple straight-line guess, “linear regression,” does very well, its predictions are closest on average, and it picks up most of the key drivers of ratings.

Feature Importance



CONCLUSION

How well did our models predict ratings?

- **Best performer: Linear Regression**
    - MAE ≈ 0.324 rating-points
    - R^2 ≈ 0.828 (explains 83% of rating variability)

      · On average, predictions are off by about one-third of a star (on a 1–5 scale).

      · Models were scored on a held-out test set to ensure they generalize beyond the training data.

- **Random Forest**
    - MAE ≈ 0.333
    - R^2 ≈ 0.810

      · Slightly more flexible but only marginally better than KNN on unseen data.

- **K-Nearest Neighbors**
    - MAE ≈ 0.348
    - R^2 ≈ 0.800

      · Simplest approach reasonable but less accurate than both RF and LR.

Evaluation approach

1. **Train/Test split**
   Reserve 20% of data as unseen “test” reviews.

2. **Metrics**

   - **MAE** (Mean Absolute Error): average distance between predicted vs. actual rating.

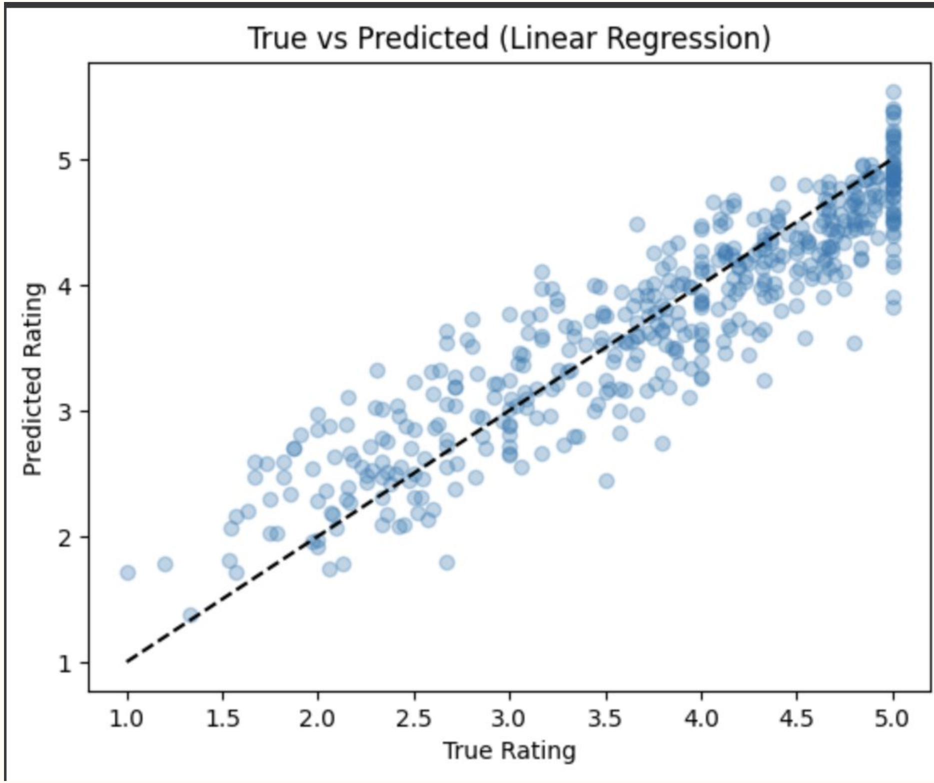
   - **R^2**: proportion of variance explained by the model.

3. **Cross-validation sanity check**
   Verified consistency of MAE/R^2 across folds to avoid lucky splits.

Grades matter most: students reward professors who give higher grades.

Words have weight: positive language in reviews strongly boosts predicted ratings.

Course context: Intro courses tend to get different baseline ratings than advanced ones.



By combining simple grade and sentiment signals with course metadata, our best model predicts professor ratings within ~0.3 stars of the true value and explains over 80% of the variation, demonstrating a strong, interpretable link between student grades, review tone, and overall satisfaction.

```
!pip install -q pandas requests tqdm
import pandas as pd, requests, json, time, os, re
from tqdm.auto import tqdm

BASE_URL = "https://planetterp.com/api/v1"
RAW_DIR = "raw_json"
os.makedirs(RAW_DIR, exist_ok=True)


limit  = 100
offset = 0
profs  = []

while True:
    resp = requests.get(f"{BASE_URL}/professors",
                        params={"limit":limit,"offset":offset})
    page = resp.json()
    if not page:
        break
    profs.extend(page)
    offset += limit
    time.sleep(0.05)

df_profs = pd.DataFrame(profs)
print(f"Total professors fetched: {len(df_profs)}")
df_profs.head(10)
```

```
import json
import pandas as pd

try:
    with open('/content/professors.json', 'r', encoding='utf-8') as f:
        all_details = json.load(f)
except json.JSONDecodeError as e:
    print(f"Error decoding JSON: {e}")
    # Print more context around the error
    print(f"Error at line {e.lineno}, column {e.colno}")
records = []
for prof in all_details:
    name = prof.get('name')
    slug = prof.get('slug')
    avg_rating = prof.get('average_rating')
    for rev in prof.get('reviews', []):
        text = rev.get('content') or rev.get('review','')
        records.append({
            'prof_name':     name,
            'slug':          slug,
            'target_rating': avg_rating,
            'course':        rev.get('course'),
            'review_text':   text.strip(),
            'grade_letter':  rev.get('grade') or rev.get('expected_grade')
        })

df_reviews = pd.DataFrame(records)
print("Total reviews loaded:", len(df_reviews))
print("Reviews with text:", df_reviews['review_text'].ne('').sum())
df_reviews.head()
```

```
Total reviews loaded: 37553
Reviews with text: 37548
```

	prof_name	slug	target_rating	course	review_text	grade_letter
0	Abani Pradhan	pradhan	3.6667	None	Pretty funny guy and he knows his stuff too. P...	B+
1	Abani Pradhan	pradhan	3.6667	NFSC112	Easy class as long as you actually watch the l...	B-
2	Abani Pradhan	pradhan	3.6667	NFSC112	Didn't learn anything, but didn't need to. Eas...	A
3	Abhijit Dasgupta	dasgupta_abhijit	4.2500	ENAE202	I took the ENAE version of this class but it i...	A-
4	Abhijit Dasgupta	dasgupta_abhijit	4.2500	ENME473	The man. Really excellent lecturer, he puts in...	A

Next steps: Generate code with df_reviews | View recommended plots | New interactive sheet

```
#map letter grades to grade points
grade_map = {
  "A+":4.0,"A":4.0,"A-":3.7, "B+":3.3,"B":3.0,"B-":2.7,
  "C+":2.3,"C":2.0,"C-":1.7, "D+":1.3,"D":1.0,"D-":0.7, "F":0.0
}
df_reviews['grade_pt'] = df_reviews['grade_letter'].map(grade_map)


#aggregate per professor
prof_agg = df_reviews.groupby(
    ['prof_name','slug','target_rating']
).agg(
    num_reviews  = ('review_text','size'),
    num_courses  = ('course','nunique'),
    avg_grade_pt = ('grade_pt','mean'),
    text_blob    = ('review_text', lambda L: " ".join(L))
).reset_index()
prof_agg.head()
```

	prof_name	slug	target_rating	num_reviews	num_courses	avg_grade_pt	text_blob
0	A Anthony	anthony	1.00	1	1	2.000000	By far the worst professor I've ever had, and ...
1	A Kruglanski	kruglanski	3.50	2	2	3.700000	DO NOT TAKE PSYC489H "Motivated Social Cogniti...
2	A Sharma	sharma_a	1.80	5	1	3.420000	Very boring, it's hard to maintain your focus ...
3	A.U. Shankar	shankar_a.u.	2.44	25	2	3.361111	Lectures are pretty dry and difficult to follo...
4	Aaron Allen	allen_aaron	5.00	1	0	NaN	Great at teaching material, very funny. If you...

Next steps: Generate code with prof_agg | View recommended plots | New interactive sheet

```
#compute NLP features
import re
from textblob import TextBlob
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')

def nlp_feats(text):
    blob = TextBlob(text)
    tokens = re.findall(r"\b\w{3,}\b", text.lower())
    return pd.Series({
        'tb_sentiment': blob.sentiment.polarity,
        'vocab_size':   len(set(tokens)),
        'avg_word_len': (sum(len(t) for t in tokens)/len(tokens)) if tokens else 0
    })

nlp_df = prof_agg['text_blob'].apply(nlp_feats)

#vADER features
sia = SentimentIntensityAnalyzer()
df_reviews['vader_compound'] = df_reviews['review_text'].apply(lambda txt: sia.polarity_scores(txt)['compound'])
vader_df = df_reviews.groupby('prof_name')['vader_compound'].mean().reset_index().rename(columns={'vader_compound':'vader_sentiment'})

df_prof = pd.concat([prof_agg.drop(columns='text_blob').reset_index(drop=True), nlp_df], axis=1)
df_prof = df_prof.merge(vader_df, on='prof_name', how='left')
df_prof.head()
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

	prof_name	slug	target_rating	num_reviews	num_courses	avg_grade_pt	tb_sentiment	vocab_size	avg_word_len	vader_sentiment
0	A Anthony	anthony	1.00	1	1	2.000000	-0.100000	43.0	5.434783	-0.493900
1	A Kruglanski	kruglanski	3.50	2	2	3.700000	0.095424	184.0	5.309735	0.134100
2	A Sharma	sharma_a	1.80	5	1	3.420000	-0.131118	130.0	5.022422	-0.537220
3	A.U. Shankar	shankar_a.u.	2.44	25	2	3.361111	0.050351	629.0	5.121762	0.168004
4	Aaron Allen	allen_aaron	5.00	1	0	NaN	0.185000	22.0	5.636364	0.874800

Next steps: Generate code with df_prof | View recommended plots | New interactive sheet

```
#drop missing target and filter >= 5 reviews
df_prof = df_prof.dropna(subset=['target_rating'])
df_prof = df_prof[df_prof['num_reviews'] >= 5].reset_index(drop=True)

#impute remaining NaNs
for c in ['avg_grade_pt','tb_sentiment','vocab_size','avg_word_len','vader_sentiment']:
    df_prof[c].fillna(df_prof[c].median(), inplace=True)

print("Cleaned professor table shape:", df_prof.shape)
df_prof.head()
```

```
Cleaned professor table shape: (1850, 10)
<ipython-input-6-d85b7ce48db4>:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df_prof[c].fillna(df_prof[c].median(), inplace=True)
```

	prof_name	slug	target_rating	num_reviews	num_courses	avg_grade_pt	tb_sentiment	vocab_size	avg_word_len	vader_sentiment
0	A Sharma	sharma_a	1.8000	5	1	3.420000	-0.131118	130.0	5.022422	-0.537220
1	A.U. Shankar	shankar_a.u.	2.4400	25	2	3.361111	0.050351	629.0	5.121762	0.168004
2	Aaron Bartlett	bartlett	2.1667	6	4	3.540000	0.141905	187.0	5.118497	0.283667
3	Aaron Finkle	finkle	3.2333	30	2	3.141379	0.093764	697.0	5.026399	0.411557
4	Aaron Kyei-asare	kyei-asare_aaron	3.4444	9	2	3.637500	0.129761	293.0	5.017241	0.438700

Next steps: Generate code with df_prof | View recommended plots | New interactive sheet

```
#how many professors do we have?
num_professors = len(all_details)
print(f"Number of professors: {num_professors}")

#how many total reviews?
num_reviews = len(df_reviews)
print(f"Number of reviews:   {num_reviews}")

#what's in the DataFrame?
print("\nDataFrame shape:", df_reviews.shape)
print("Columns:", list(df_reviews.columns))

df_reviews.head()
```

```
Number of professors: 13427
Number of reviews:   37553

DataFrame shape: (37553, 8)
Columns: ['prof_name', 'slug', 'target_rating', 'course', 'review_text', 'grade_letter', 'grade_pt', 'vader_compound']
```

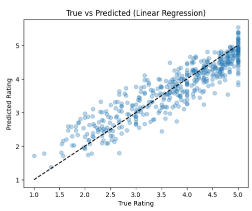
	prof_name	slug	target_rating	course	review_text	grade_letter	grade_pt	vader_compound
0	Abani Pradhan	pradhan	3.6667	None	Pretty funny guy and he knows his stuff too. P...	B+	3.3	0.9413
1	Abani Pradhan	pradhan	3.6667	NFSC112	Easy class as long as you actually watch the l...	B-	2.7	-0.2396
2	Abani Pradhan	pradhan	3.6667	NFSC112	Didn't learn anything, but didn't need to. Eas...	A	4.0	-0.5732

Next steps: Generate code with df_reviews View recommended plots New interactive sheet

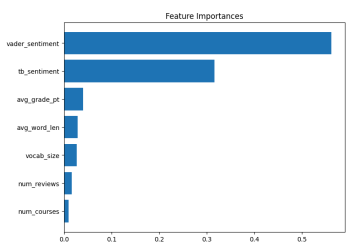
```
#train/Test split and modeling
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score

#features and target
feature_cols = ['num_reviews','num_courses','avg_grade_pt','tb_sentiment','vocab_size','avg_word_len','vader_se
X = df_prof[feature_cols]
y = df_prof['target_rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

models = {
    'KNN': Pipeline([('impute', SimpleImputer(strategy='median')), ('scale', StandardScaler()), ('knn', KNeighb
    'RF': Pipeline([('impute', SimpleImputer(strategy='median')), ('scale', StandardScaler()), ('rf', RandomFor
    'LR': Pipeline([('impute', SimpleImputer(strategy='median')), ('scale', StandardScaler()), ('lr', LinearReg
}
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    results.append((name, mean_absolute_error(y_test, preds), r2_score(y_test, preds)))
    print(f"{name}: MAE={results[-1][1]:.3f}, R2={results[-1][2]:.3f}")
```

```
KNN: MAE=0.348, R2=0.800
 RF: MAE=0.333, R2=0.810
 LR: MAE=0.324, R2=0.828
```

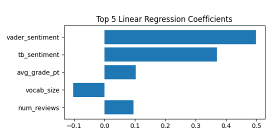
```
import matplotlib.pyplot as plt

preds = models['LR'].predict(X_test)
plt.scatter(y_test, preds, alpha=0.3)
plt.plot([1,5],[1,5], 'k--')
plt.xlabel("True Rating")
plt.ylabel("Predicted Rating")
plt.title("True vs Predicted (Linear Regression)")
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

model = models['RF'].named_steps['rf']
feature_names = X_train.columns

importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(8, 6))
plt.barh([feature_names[i] for i in indices],
         importances[indices],
         align='center')
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.gca().invert_yaxis()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

lr_model = models['LR'].named_steps['lr']
coefs = lr_model.coef_
features = X_train.columns

idx = np.argsort(np.abs(coefs))[::-1]

plt.figure(figsize=(6,3))
plt.barh(features[idx][:5], coefs[idx][:5])
plt.title("Top 5 Linear Regression Coefficients")
plt.xlabel("Coefficient value")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```




```
from sklearn.inspection import permutation_importance

perm = permutation_importance(
    models['KNN'], X_test, y_test,
    n_repeats=10,
    random_state=42,
    n_jobs=-1
)

importances = perm.importances_mean
features = X_test.columns
idx = np.argsort(importances)[::-1]

plt.figure(figsize=(6,3))
plt.bar(features[idx][:5], importances[idx][:5])
plt.title("Top 5 KNN Permutation Importances")
plt.ylabel("Decrease in R² when permuted")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```