National College of Ireland

# Configuration Manual

MSc Research Project
MSc. Data Analytics

Abhinandan Nahar
Student ID: X22202871

School of Computing
National College of Ireland

Supervisor: Prof. David Hamil

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Abhinandan Anil Nahar |
| **Student ID:** | X22202871 |
| **Programme:** | MSc. Data Analytics **Year:** 2023 - 2024 |
| **Module:** | MSc Research Project |
| **Supervisor:** | David Hamil |
| **Submission Due Date:** | 12-08-2024 |
| **Project Title:** | Configuration Manual |

**Word Count:** 265 **Page Count** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Abhinandan Nahar

**Date:** 12-08-2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Abhinandan Nahar
X22202871

# 1. Introduction

This Configuration Manual provides detailed instructions for setting up, configuring, and maintaining the social media-like application built with Django. It is intended for system administrators, developers, and IT professionals responsible for deploying and managing the application. The application is a Twitter-like platform that allows users to post tweets, follow other users, and explore content based on interests. It features user authentication, tweet clustering, and personalized content recommendations using machine learning techniques.

# 2. Prerequisites

Required software and libraries

- Python 3.8 or higher
- Django 3.2 or higher
- PostgreSQL 12 or higher
- OpenAI API account and key
- Additional Python libraries: numpy, pandas, scikit-learn, matplotlib, seaborn, nltk

# 3. Installation

**Setting up the development environment**

I. Install Python from https://www.python.org/downloads/
II. Create a virtual environment:
*III.* *python -m venv venv*
*IV.* *source venv/bin/activate  # On Windows use: venv\Scripts\activate*

**Installing dependencies**

Install the required Python packages:

*pip install django psycopg2-binary openai numpy pandas scikit-learn matplotlib seaborn nltk*

# 4. Configuration

**Static files configuration**

Configure static files in `settings.py`:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC_URL = 'static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]

STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

**Authentication settings**

Add the custom user model to `settings.py`:

```
AUTH_USER_MODEL = 'tweets.User'

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'tweets'
]
```

**OpenAI API configuration**

Store your OpenAI API key securely (e.g., in an environment variable) and configure it in your application:

```
def generate_interest_scores(tweet_text, interest_topics):
    """Generate interest scores using GPT-3.5 API."""
    prompt = (
        f"Analyze the following tweet text and assign a float score between 0 and 1 for each "
        f"interest domain: {', '.join(interest_topics)}.\n\n"
        f"Tweet: {tweet_text}\n\n"
        f"Scores:"
    )
    response = openai.Completion.create(
        engine='text-davinci-003',
        prompt=prompt,
        max_tokens=100,
        temperature=0.5
```

# 5. Models

### 5.1 User model

The User model extends Django's AbstractUser and includes additional fields for interests and following relationships.

```
class User(AbstractUser):
    date_of_birth = models.DateField(null=True, blank=True)
    interests = models.JSONField(default=dict)
    following = models.ManyToManyField('self', symmetrical=False, related_name='followers')


    user_permissions = models.ManyToManyField(
        Permission,
        verbose_name='user permissions',
        blank=True,
        related_name='custom_user_set',
        related_query_name='custom_user',
    )
    groups = models.ManyToManyField(
        Group,
        verbose_name='groups',
        blank=True,
        related_name='custom_user_set',
        related_query_name='custom_user',
    )

    def __str__(self):
        return self.username
```

### 5.2 Tweet model

The Tweet model represents user posts and includes fields for text content, creation time, interest scores, and likes.

```python
class Tweet(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='tweets')
    text = models.CharField(max_length=280)
    created_at = models.DateTimeField(auto_now_add=True)
    interest_scores = models.JSONField(default=dict, blank=True)
    likes = models.ManyToManyField(User, related_name='liked_tweets', blank=True)

    def __str__(self):
        return f"Tweet by {self.user.username} on {self.created_at}"

    def like_count(self):
        return self.likes.count()

    def save(self, *args, **kwargs):
        if not self.pk:
            self.generate_interest_scores()
        super().save(*args, **kwargs)
```

### 6. Views and URL Configuration

#### Authentication views

Implement views for user registration, login, and logout. Configure URLs in `urls.py`:

```python
def login_view(request):
    if request.method == 'POST':
        form = CustomAuthenticationForm(data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('feed')
    else:
        form = CustomAuthenticationForm()
    return render(request, 'tweets/login.html', {'form': form})

def logout_view(request):
    logout(request)
    return redirect('login')
```

### Tweet-related views

```python
@login_required
def feed(request):
    user = request.user
    user_interests = user.interests

    # Fetch all tweets and annotate them with a custom score field
    tweets = Tweet.objects.annotate(
        custom_score=Value(0.0, output_field=FloatField())
    ).all()

    # Calculate custom score for each tweet
    scored_tweets = []
    for tweet in tweets:
        score = 0
        for interest, user_score in user_interests.items():
            tweet_score = tweet.interest_scores.get(interest, 0)
            score += user_score * tweet_score

        scored_tweets.append({
            'tweet': tweet,
            'score': score
        })

    # Sort tweets by custom score (descending order)
    sorted_tweets = sorted(scored_tweets, key=itemgetter('score'), reverse=True)

    # Extract just the tweet objects for the template
    sorted_tweet_objects = [item['tweet'] for item in sorted_tweets]

    return render(request, 'tweets/feed.html', {'tweets': sorted_tweet_objects})
```

```python
@login_required
@transaction.atomic
def like_tweet(request, tweet_id):
    tweet = get_object_or_404(Tweet, id=tweet_id)
    user = request.user

    if tweet.likes.filter(id=user.id).exists():
        # Unlike the tweet
        tweet.likes.remove(user)
        adjustment_factor = -0.01  # Slight decrease when unliking
    else:
        # Like the tweet
        tweet.likes.add(user)
        adjustment_factor = 0.02  # Slight increase when liking

    # Update user's interest scores
    user_interests = user.interests
    tweet_interests = tweet.interest_scores

    for category, tweet_score in tweet_interests.items():
        if category in user_interests:
            # Adjust the user's interest score
            new_score = user_interests[category] + (tweet_score * adjustment_factor)
            # Ensure the score stays within 0-1 range
            user_interests[category] = max(0, min(1, new_score))
        else:
            # If the user doesn't have this category, add it with a small initial value
            user_interests[category] = tweet_score * adjustment_factor

    # Update the user's interests
    User.objects.filter(id=user.id).update(interests=user_interests)
```

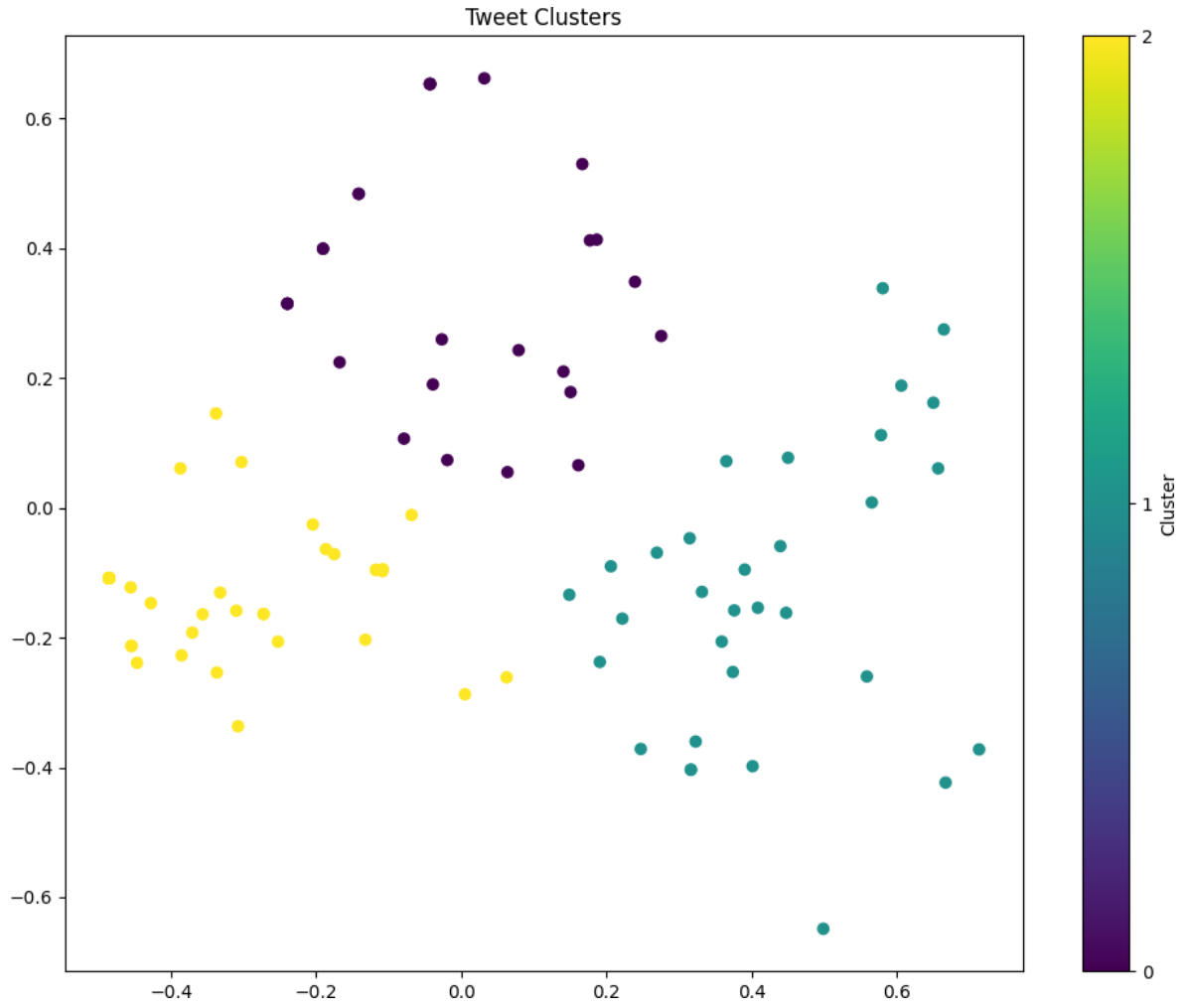
**User profile and connections views**

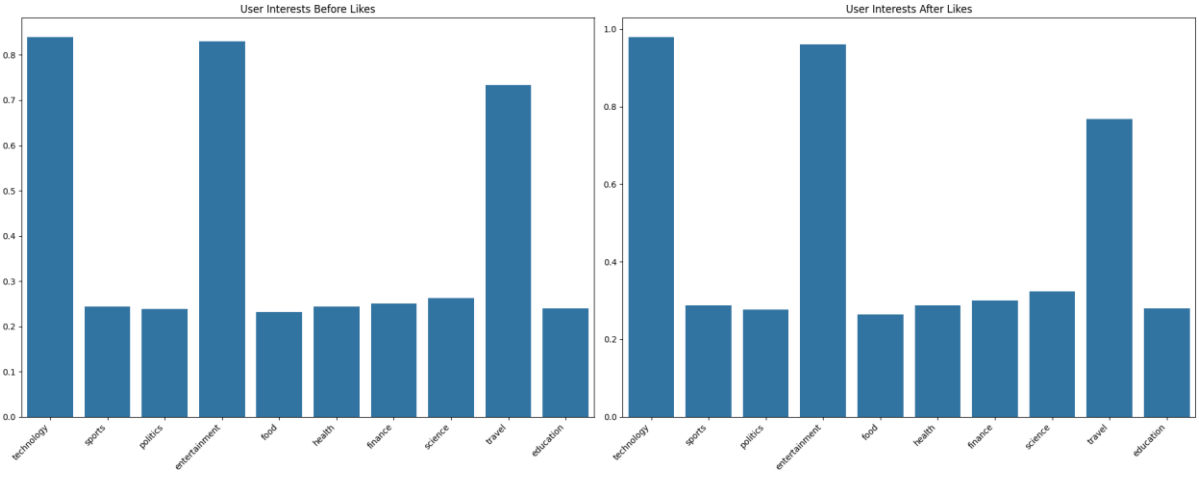
```python
@login_required
def connections(request):
    user = request.user
    following = user.following.all()
    followers = User.objects.filter(following=user)
    return render(request, 'tweets/connections.html', {
        'following': following,
        'followers': followers,
    })

def register(request):
    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('choose_interests')  # Redirect to the
    else:
        form = CustomUserCreationForm()
    return render(request, 'tweets/register.html', {'form': form})
```

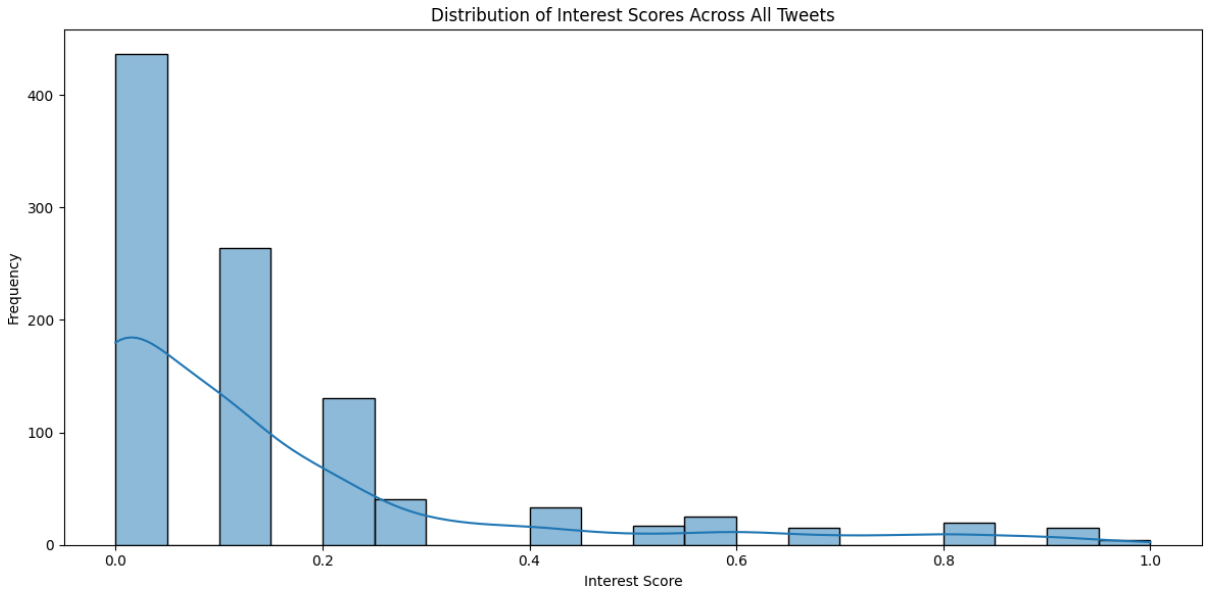
## Exploration and clustering views

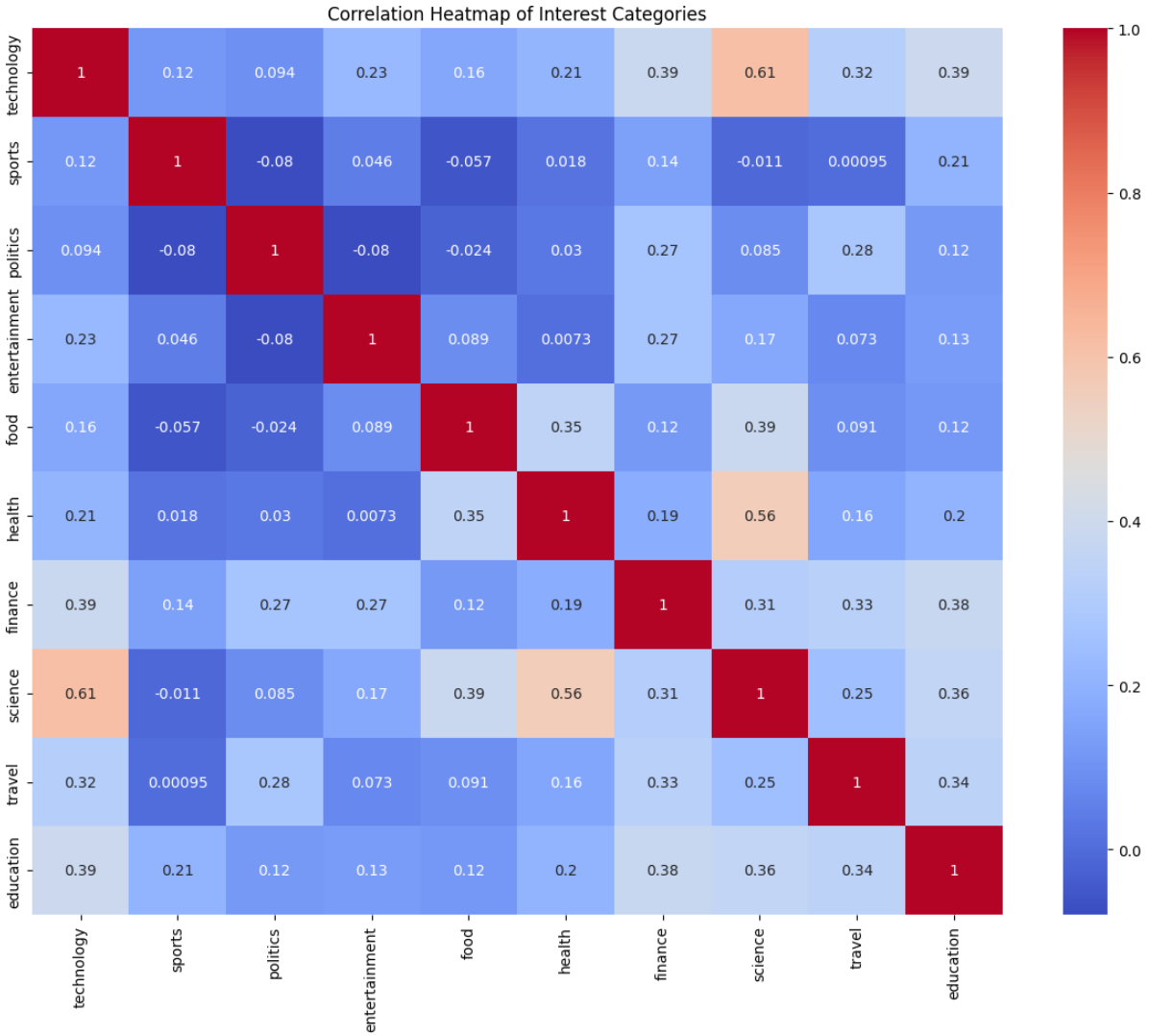
```python
@login_required
def explore(request):
    if request.method == 'POST':
        n_clusters = int(request.POST.get('n_clusters', 5))  # Default to 5 if not specified
        return redirect('show_clusters', n_clusters=n_clusters)
    return render(request, 'tweets/explore.html')


@login_required
def show_clusters(request, n_clusters):
    tweets = Tweet.objects.all()
    n_samples = tweets.count()

    n_clusters = min(n_clusters, n_samples)

    if n_samples == 0:
        return render(request, 'tweets/show_clusters.html', {'error': 'No tweets available for

    X = []
    for tweet in tweets:
        X.append([tweet.interest_scores.get(interest, 0) for interest in INTEREST_CHOICES])
    X = np.array(X)

    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    labels = kmeans.fit_predict(X)

    # Group tweets by cluster
    clusters = {}
    for tweet, label in zip(tweets, labels):
        if label not in clusters:
            clusters[label] = []
        clusters[label].append(tweet.id)
```

# 7. Analysis


Tweet Clusters
Cluster
0
1
2
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6


User Interests Before Likes
User Interests After Likes
technology
sports
politics
entertainment
food
health
finance
science
travel
education


Distribution of Interest Scores Across All Tweets
Frequency
Interest Score
0
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0


Correlation Heatmap of Interest Categories
technology 1 0.12 0.094 0.23 0.16 0.21 0.39 0.61 0.32 0.39
sports 0.12 1 -0.08 0.046 -0.057 0.018 0.14 -0.011 0.00095 0.21
politics 0.094 -0.08 1 -0.08 -0.024 0.03 0.27 0.085 0.28 0.12
entertainment 0.23 0.046 -0.08 1 0.089 0.0073 0.27 0.17 0.073 0.13
food 0.16 -0.057 -0.024 0.089 1 0.35 0.12 0.39 0.091 0.12
health 0.21 0.018 0.03 0.0073 0.35 1 0.19 0.56 0.16 0.2
finance 0.39 0.14 0.27 0.27 0.12 0.19 1 0.31 0.33 0.38
science 0.61 -0.011 0.085 0.17 0.39 0.56 0.31 1 0.25 0.36
travel 0.32 0.00095 0.28 0.073 0.091 0.16 0.33 0.25 1 0.34
education 0.39 0.21 0.12 0.13 0.12 0.2 0.38 0.36 0.34 1
technology
sports
politics
entertainment
food
health
finance
science
travel
education
1.0
0.8
0.6
0.4
0.2
0.0