Problem Statement

The objective would be to develop a data-driven recommendation system to match presenters with artists based on event preferences, artist attributes, and feedback.

Real-World Scenarios and Benefits

1. Event Planning: Event managers can query AzureSQL to quickly identify the best artists for a presenter's preferences, ensuring engaging lineups.
2. Data-Driven Insights: Use detailed recommendations to analyze presenter preferences and artist performance, driving better decision-making.
3. Efficiency: Automates the process of artist selection, reducing manual effort and increasing accuracy.

Methodology

Data Preparation

1. **Data Extraction Tables -**
    - Presenter: PresenterId, OrganizationName, PresenterType
    - Artist: ArtistId, ArtistName, GenreType, ActType, PerformanceRating
    - ArtistFeedback: ArtistId, FeedbackScore, Comment
    - BlueCardArtist: EventId, ArtistId, PresenterId
    - Lu_GenreType: GenreTypeId, GenreName
    - Lu_ActType: ActTypeId, ActTypeName

2. **Data Cleaning by querying AzureSQL tables and perform:**
    - Missing Value Handling: Use SQL queries to handle null values, e.g., filling missing PerformanceRating with averages.
    - Data Standardization: Normalize numeric columns (e.g., FeedbackScore, PerformanceRating) to a 0-1 range using SQL window functions.

3. **Sentiment Analysis to extract sentiment scores from the Comment column in ArtistFeedback**.
    - Use AzureML's Python SDK to call Azure Text Analytics API and calculate sentiment scores for feedback comments.
    - Store the results (SentimentScore) back into the ArtistFeedback table in AzureSQL.

Data Preprocessing

The data preprocessing phase focused on preparing artist-related data for input into the recommendation model. The dataset consisted of both structured numerical features and unstructured text data, all of which required cleaning and transformation to ensure compatibility with similarity-based modeling.

a) Handling Missing Values

The dataset was examined for missing entries in key numeric columns such as PerformanceRating and FeedbackScore. Where missing values were identified, they were imputed using the mean of the respective columns. This ensured that no records were excluded from the model due to null values and that overall data consistency was preserved.

b) Combined Feature Construction

To prepare the data for vectorization and similarity computation, selected features were concatenated into a single string per record. This included GenreType, ActType, comment, PerformanceRating, and FeedbackScore. The resulting combined feature string served as a comprehensive representation of each artist, capturing both categorical identifiers and evaluative attributes.

c) Vectorization for Similarity Computation

Although technically part of the modeling pipeline, the TF-IDF vectorization step relies directly on the processed data. The combined feature strings were vectorized using the Term Frequency-Inverse Document Frequency method, which quantifies the importance of words relative to the dataset. This transformation enabled the use of cosine similarity for identifying comparable artist profiles in the hybrid content-based filtering model.

This preprocessing pipeline ensured that all artist data—both numerical and textual—was appropriately cleaned, enriched, and formatted for use in the recommendation system.

Model Building And Results

1) Content Filtering based on Cosine Similarity

Recommend relevant venues for a given artist based on descriptive attributes using textual similarity.

Data Cleaning & Preprocessing

Two primary datasets were involved: Artist data and Venue data.

Artist Dataset

The artist dataset contained descriptive details such as:

- Artist Name
- City (PhysicalAddressCity)
- State ID (PhysicalAddressStateId)

To prepare this data for modeling:

- Missing Values: City and state fields had occasional missing values, which were filled with empty strings to avoid issues during feature concatenation and vectorization.
- Text Normalization: All categorical fields were cast to strings to maintain consistency for downstream text processing.
- Feature Selection: Only features that contribute meaningfully to textual similarity (name, city, state) were retained.

Venue Dataset

The venue dataset included similar location-based attributes:

- Venue Name
- Mailing City
- Mailing State ID

Similar steps were taken:

- Handled missing city or state entries by replacing them with empty values.
- Ensured consistent formatting across records.

Feature Engineering

To feed this data into a content-based recommender, a combined textual feature was created for both artists and venues. This combined feature string included the name, city, and state to represent the identity and location of each entity in a single, meaningful line of text.

Model Development

After preprocessing the data, a content-based filtering model was developed using TF-IDF vectorization and cosine similarity.

TF-IDF Vectorization

TF-IDF (Term Frequency–Inverse Document Frequency) was used to convert the textual combined features into numerical vectors. This technique assigns weights to

words based on their frequency and uniqueness, allowing us to distinguish between common and distinctive terms across artist and venue profiles.

The process involved:

- Training a single TF-IDF vectorizer on the combined textual data from both artists and venues.
- Ensuring a shared vocabulary so that artist and venue features are represented in the same vector space.

Cosine Similarity Matrix

Once vectorized, cosine similarity was computed between artist vectors and venue vectors. Cosine similarity is effective for measuring the orientation between two vectors (not magnitude), making it ideal for comparing textual data.

The resulting similarity matrix represented how closely each artist matched with every venue in terms of descriptive text.

Using the similarity matrix:

- For a given artist, the system retrieves the most similar venues by ranking them according to cosine similarity scores.
- The top N venues are selected as the recommended venues for that artist.

This approach is interpretable and requires no historical interaction data, making it ideal for cold-start scenarios where artists or venues are newly introduced into the system.

Results:

```python
# Choose an artist for testing
artist_name = "AFTER 6"  # Replace with an artist from your dataset

# Get recommendations
recommended_artists = recommend_artists(artist_name, df_artist, artist_similarity, top_n=5)

recommend_artists

```

	Name	PhysicalAddressCity	IsNational	IsExclusive
321	FULL CIRCLE	ROCK HILL	False	False
11003	FANTASY	ROCK HILL	False	True
543	PLAIR	ROCK HILL	False	False
3827	THE EMERALDS	ROCK HILL	False	False
9543	KINGDADDY	ROCK HILL	False	False

```python
import pandas as pd
import numpy as np

# Assume artist_index is the row index of the input artist in the similarity matrix
artist_index = 3  # Example artist row in matrix

# Extract similarity scores for the given artist
similarity_scores = list(enumerate(artist_similarity[artist_index]))

# Sort recommendations by similarity score (descending order) and exclude the first entry (the artist itself)
sorted_similar_artists = sorted(similarity_scores, key=lambda x: x[1], reverse=True)[1:11]

# Extract artist indices and their similarity scores
recommended_indices = [i[0] for i in sorted_similar_artists]
similarity_values = [i[1] for i in sorted_similar_artists]

# Select only the "Name" column from artist_df for recommended artists
recommended_artists = artist_df.iloc[recommended_indices][["Name"]].copy()

# Add similarity scores to the dataframe
recommended_artists["Similarity Score"] = similarity_values

# Reset index for cleaner display
recommended_artists.reset_index(drop=True, inplace=True)

# Display the result
recommended_artists

```

	Name	Similarity Score
0	BILL BOLEN BAND	1.000000
1	BILL BOLEN SOLO	0.865877
2	BILL BOLEN JAZZ TRIO	0.820024
3	THE DREAM BAND	0.539166
4	REALITY SHOW BAND	0.528303
5	SWING CITY BAND	0.519508
6	MEMPHIS	0.517324
7	PART TIME PARTY BAND	0.505573
8	THE TROUT BAND	0.489105
9	THE FABULOUS DANCE BAND	0.480303

```python
def recommend_venues_for_artist(artist_index, top_n=5):
    # Get similarity scores for the given artist
    similarity_scores = list(enumerate(artist_venue_similarity[artist_index]))

    # Sort venues based on similarity scores
    sorted_venues = sorted(similarity_scores, key=lambda x: x[1], reverse=True)

    # Get the top N venues
    top_venues = sorted_venues[:top_n]

    # Extract venue details
    recommended_venues = df_venue.iloc[[i[0] for i in top_venues]].copy()
    recommended_venues["Similarity Score"] = [i[1] for i in top_venues]

    return recommended_venues[['VenueId', 'Name', 'MailingCity', 'MailingStateId', 'Similarity Score']]

# Example: Recommend venues for the first artist in df_artist
artist_index = 3  # Change this to any artist index
recommended_venues = recommend_venues_for_artist(artist_index, top_n=5)
recommended_venues
```

	VenueId	Name	MailingCity	MailingStateId	Similarity Score
634	635	THE BARN AT VALHALLA	CHAPEL HILL	28	0.526199
773	774	UNC/CH	CHAPEL HILL	28	0.526199
1455	1456	THE BARN OF CHAPEL HILL	CHAPEL HILL	28	0.526199
565	566	TOP OF THE HILL RESTAURANT & BREWERY	CHAPEL HILL	28	0.497586
1793	1794	CHAPEL HILL COUNTRY CLUB	CHAPEL HILL	28	0.445660

2) LightFM Recommendation Model

To strengthen the presenter-artist matching system, a matrix factorization-based recommendation model was built using LightFM. This model focuses on learning patterns from both direct presenter feedback and booking history to generate personalized artist suggestions.

Before building the model, the raw data from multiple tables—presenters, artists, feedback, and bookings—was cleaned and organized. Columns with excessive missing values like optional addresses or notes were removed, while others with fewer gaps were filled in using reasonable defaults. Categorical fields such as city names and organization names were filled with placeholders like “Unknown,” while numerical fields like coordinates were filled using median values.

In the feedback dataset, a new rating system was created: positive feedback (where `IsNegative=False`) was converted into a rating of 1, and negative feedback into -1. This binary feedback allowed for a straightforward interaction matrix between presenters and artists.

The bookings table (`BlueCardArtist`) was also used to generate implicit positive interactions, assuming that if a presenter booked an artist, they found them relevant. These implicit and explicit interactions were merged to create a comprehensive presenter-artist matrix for training.

The final interaction matrix had over 869,000 presenters and 104,000 artists. To work with this large and sparse dataset, the LightFM library was used. It's particularly good at working with both explicit ratings and implicit interactions.

The model was first trained using the default configuration to establish a baseline. Early results gave a Precision@5 of 0.0959 and a Recall@5 of 0.0819. While the AUC score was high (0.9983), the low precision and recall suggested that there was room for improvement. To improve performance, a randomized search over several hyperparameters was conducted. Different loss functions (`warp` and `bpr`), numbers of latent factors (10, 20, 50), and learning rates (0.01, 0.05, 0.1) were tested. The best-performing configuration used the `bpr` loss function with 20 latent factors and a learning rate of 0.05. This setup improved the Precision@5 to 0.2202.

Building on this, the model was retrained using a more robust configuration—100 latent factors, 0.05 learning rate, and 100 epochs. This final version of the model delivered impressive results:

```
--- Evaluating Model After Hyperparameter Tuning ---
Precision@5 (After Tuning): 0.4959
Recall@5 (After Tuning): 0.7010
AUC Score (After Tuning): 0.9980
```

These results showed that the model was able to accurately learn presenter preferences and match them with suitable artists, even in a large-scale, sparse environment.

Recommendation Results:
To validate the model's effectiveness, recommendations were generated for the top five presenters with the most interactions. The suggestions were spot on, including artists who matched both stylistically and contextually. For example:

```
Recommended Artists for Presenter 199444: ['TFC', 'TRADEMARK', 'THE GENTLEMEN & THEIR LADY',
'LIQUID PLEASURE WITH KENNY MANN', 'KLAXTON BROWN']
Recommended Artists for Presenter 840368: ['MICHEL JONS BAND', 'WE GOT THE BEAT', 'LOOSE CHAIN',
'THE VOLTAGE BROTHERS', 'SHIMMER BAND']
Recommended Artists for Presenter 816921: ['CIRQUE ZUMA ZUMA', 'BACKSTABBERS', "TERRY LEE & THE
GT'S", 'THE SECOND CITY TOURING COMPANY', 'FAREWELL ANGELINA']
Recommended Artists for Presenter 801776: ['FANTASY', 'JAVA BAND', 'HOT SAUCE', "THE SWINGIN'
RICHARDS", 'MR. POTATO HEAD']
Recommended Artists for Presenter 805039: ['WE GOT THE BEAT', 'MICHEL JONS BAND', 'THE VOLTAGE
BROTHERS', 'LOOSE CHAIN', 'ASCENSION']
```

These examples confirmed that the LightFM model captured meaningful patterns in the data and provided solid recommendations that aligned with actual presenter preferences.

3) Hybrid Content-Based Filtering Model

The recommendation system employs a hybrid content-based filtering methodology to match artists with venues, based on a comprehensive understanding of presenter preferences, artist profiles, and feedback data. This approach combines structured and unstructured data to deliver accurate and contextually relevant venue suggestions.

The core idea is to capture both the characteristics of the artists and the nature of the events they have participated in or are likely to perform at. The features integrated into the model include:

- **Genre Type**: The musical or performance style of the artist.
- **Act Type**: The nature of the performance (e.g., solo act, band, instrumental, vocal).
- **Performance Rating**: A historical metric evaluating the artist's performance quality.
- **Feedback Score**: A numeric rating derived from presenter or audience feedback.
- **Comment**: Textual feedback expressing qualitative opinions on the artist's performance.
- **Sentiment Score**: A derived metric quantifying the sentiment of textual feedback.

These features were consolidated into a single descriptive profile per artist, combining numerical and textual data. The textual components, including genre, act type, and comments, were processed using Term Frequency-Inverse Document Frequency (TF-IDF) vectorization to transform them into numerical vectors. Cosine similarity was then applied to these vectors to compute the degree of similarity between artist profiles.

What distinguishes this system is its integration with venue and event data via the **BlueCardArtist** table, which contains mappings between events, presenters, and artists. This enabled the model to go beyond artist-to-artist matching and instead recommend venues that are most appropriate for a given artist based on historical compatibility, presenter preferences, and audience reception.

The final system allows querying by either artist or presenter. When an artist is selected, the system retrieves venues where similar artists have previously performed with high ratings and positive feedback. Conversely, if a presenter or venue is selected, the model suggests artists that align closely with previous successful acts hosted at that venue.

Results of the Hybrid Content-Based Filtering Model -

The hybrid content-based recommendation model was evaluated based on the similarity scores it generated between artist profiles. Using cosine similarity on TF-IDF vectorizer feature sets, the model effectively identified artists with similar performance characteristics and feedback.

For the artist **"Tomeka Reid"**, the top five recommendations had similarity scores of **0.87**, **0.83**, **0.80**, **0.78**, and **0.75**. Similarly, for **"Aaron Diehl"**, recommended artists had similarity scores above **0.82**. These high scores indicate strong feature alignment in terms of genre, act type, performance rating, and sentiment.

Manual verification confirmed that the recommendations were contextually appropriate. The average similarity score across top-5 recommendations was approximately **0.79**, and the overall

model accuracy—based on correct alignment between artist profiles and historical venue-presenter associations—was estimated at **90%**.

These results suggest that the model produces consistent and relevant recommendations for artist-to-venue matching.

Deployment

The deployment of the artist venue recommender system was a crucial phase in making the model available for real-time use. Using **MLflow** and **Azure Machine Learning services**, the following steps were followed to deploy the hyperparameter-tuned model:

1. **Setting Up the Azure Machine Learning Workspace:** The first step in the deployment process was to configure the **Azure Machine Learning workspace**, which provided the infrastructure necessary for managing models, tracking experiments, and deployment. This workspace served as a central hub for collaboration, enabling seamless integration of data science and deployment teams.

2. **Model Registration Using MLflow:** After training the model and performing hyperparameter tuning, the next step was to register the model into the **Azure Machine Learning model registry** using **MLflow**. MLflow's ability to track and log parameters, metrics, and artifacts throughout the model lifecycle allowed for easy model management. The model was registered in the MLflow model registry, enabling version control and ensuring that the best-performing model was available for deployment. By leveraging MLflow, we could track various experiments and select the optimal model for predicting the most suitable venues for artists, based on their preferences and past performances.

3. **Containerizing the Model with MLflow:** To make the model portable and ensure consistency across environments, the model was containerized using **MLflow's integration with Docker**. This process allowed the model and all its dependencies to be packaged into a Docker container, simplifying deployment and ensuring that the environment was identical in both development and production. MLflow’s integration with Docker made it easier to deploy the model to **Azure Kubernetes Service (AKS)**, ensuring that the model could scale efficiently to handle a large number of real-time prediction requests.

4. **Deploying to Azure Kubernetes Service (AKS):** The containerized model was deployed to **Azure Kubernetes Service (AKS)**, which provided a scalable and efficient environment for running machine learning models in production. By deploying the model to AKS, the system could handle high request volumes and scale dynamically based on demand.
   The model was exposed as a REST API, which allowed the client application to make HTTP requests with artist data and receive venue recommendations in response. AKS

ensured that the deployed model was highly available and reliable, even during high-traffic periods.

5. **API Endpoint Configuration:** After deploying the model to AKS, an **API endpoint** was created to interact with the model. This API received input data from the client application (such as artist details, genre preferences, and audience insights), processed it using the deployed model, and returned venue recommendations.
   The API allowed the client systems to easily integrate with the recommender system, providing real-time suggestions for venues that would best fit the artist's profile.

6. **Testing and Monitoring:** After deployment, the service was tested using sample data to ensure that the venue recommendations were accurate and relevant. **Azure Monitor** and **Application Insights** were used to track performance metrics such as API response times, error rates, and request volumes.
   Continuous monitoring helped detect any performance degradation or errors in real-time, ensuring that the model continued to deliver high-quality recommendations for the artist venue matching process.