RESTAURANT RECOMMENDATION SYSTEM

ZENAB OSAMA 2305289
MARIAM AHMED 2305300
AHMED SADEK 2305355

1-introduction:

This report provides an overview of a restaurant recommendation system developed using Python. The system processes restaurant data, filters it based on user preferences, and generates personalized recommendations. The report includes code snippets and explanations for each component.

2. Data Processing Module

Function: preprocess_data()
Purpose:
Loads and cleans the restaurant dataset, handling missing values, normalizing cuisines, and categorizing price ranges.

```python
def load_data(filepath):
    """Load the dataset from CSV file"""
    try:
        df = pd.read_csv(filepath)
        print("Data loaded successfully")
        return df
    except FileNotFoundError:
        print("File not found. Using synthetic data instead.")
        return create_synthetic_data()
```

This function attempts to load restaurant data from a CSV file. If the file isn't found, it falls back to synthetic data generation. This makes the system more robust by ensuring it always has data to work with.

```python
def create_synthetic_data():
    """Create synthetic data if real data is not available"""
    np.random.seed(42)
    num_restaurants = 500
    cuisines = ['Indian', 'Chinese', 'Italian', 'Mexican', 'Japanese', 'Thai', 'American', 'Medi
terranean']
    locations = ['Downtown', 'Midtown', 'Uptown', 'Eastside', 'Westside']

    data = {
        'Restaurant Name': [f"Restaurant {i}" for i in range(num_restaurants)],
        'Cuisines': [', '.join(np.random.choice(cuisines, np.random.randint(1, 3), replace=Fals
e)) for _ in range(num_restaurants)],
        'Location': np.random.choice(locations, num_restaurants),
        'Average Cost for two': np.random.randint(200, 1500, num_restaurants),
        'Aggregate rating': np.round(np.random.uniform(2.5, 5, num_restaurants), 1),
        'Votes': np.random.randint(10, 5000, num_restaurants),
        'Latitude': np.random.uniform(12.8, 13.2, num_restaurants),
        'Longitude': np.random.uniform(77.5, 77.7, num_restaurants)
    }
    return pd.DataFrame(data)
```

Generates realistic synthetic restaurant data with random attributes when real data isn't available. The data includes names, cuisines, locations, prices, ratings, votes, and geographic coordinates. The fixed random seed ensures reproducibility.

```python
def clean_data(df):
    """Clean and preprocess the data"""
    # Remove duplicates
    df = df.drop_duplicates(subset=['Restaurant Name', 'Location'])

    # Normalize cuisines
    df['Cuisines'] = df['Cuisines'].str.lower().str.strip()

    # Handle missing values
    df = df.dropna(subset=['Cuisines', 'Average Cost for two', 'Aggregate rating'])

    # Create price bucket
    df['Price Bucket'] = pd.cut(
        df['Average Cost for two'],
        bins=[0, 400, 700, float('inf')],
        labels=['Low', 'Medium', 'High']
    )

    # Extract primary cuisine
    df['Primary Cuisine'] = df['Cuisines'].str.split(',').str[0].str.strip()

    return df
```

Performs essential data cleaning including:

- Removing duplicate restaurants
- Normalizing cuisine names
- Handling missing values
- Creating price categories
- Extracting primary cuisine for simpler filtering

```python
def preprocess_data(filepath='zomato_data.csv'):
    """Main data processing function"""
    df = load_data(filepath)
    df = clean_data(df)
    return df
```

The main function that orchestrates the data loading and cleaning process

3. Recommendation Engine

PROVIDES METHODS TO FILTER RESTAURANTS BASED ON USER PREFERENCES AND CALCULATE RECOMMENDATION SCORES.

```
class RestaurantRecommender:
    def __init__(self, data):
        self.data = data
        self.scaler = MinMaxScaler()
```

Initializes the recommender with restaurant data and a scaler for normalizing rating and vote values.

```python
def filter_restaurants(self, cuisine=None, location=None, price_bucket=None):
    """Filter restaurants based on user preferences"""
    filtered = self.data.copy()

    if cuisine:
        filtered = filtered[filtered['Cuisines'].str.contains(cuisine.lower())]

    if location:
        filtered = filtered[filtered['Location'].str.lower() == location.lower()]

    if price_bucket:
        filtered = filtered[filtered['Price Bucket'] == price_bucket]

    return filtered
```

Filters restaurants based on user preferences for cuisine, location, and price range. Uses case-insensitive matching for better user experience.

```python
def calculate_scores(self, df):
    """Calculate recommendation scores"""
    # Normalize ratings and votes
    df['Rating_norm'] = self.scaler.fit_transform(df[['Aggregate rating']])
    df['Votes_norm'] = self.scaler.fit_transform(df[['Votes']])

    # Calculate weighted score (70% rating, 30% votes)
    df['Score'] = 0.7 * df['Rating_norm'] + 0.3 * df['Votes_norm']
    return df
```

Creates a composite score for each restaurant by:

1. Normalizing ratings and vote counts
2. Combining them with weights (70% rating, 30% popularity)

```python
    def recommend(self, cuisine=None, location=None, price_bucket=None, top_n=5):
        """Generate recommendations based on filters"""
        filtered = self.filter_restaurants(cuisine, location, price_bucket)

        if filtered.empty:
            return pd.DataFrame()

        scored = self.calculate_scores(filtered)
        recommendations = scored.sort_values('Score', ascending=False).head(top_n)

        return recommendations
```

The main recommendation function that:

1. Filters restaurants based on user criteria
2. Calculates scores for filtered restaurants
3. Returns top N recommendations sorted by score

```python
def generate_explanation(self, restaurant):
    """Generate explanation for recommendation"""
    explanation = f"Matched on {restaurant['Primary Cuisine'].title()} cuisine"

    if pd.notna(restaurant['Price Bucket']):
        explanation += f", {restaurant['Price Bucket']} budget"

    explanation += f" with {restaurant['Aggregate rating']} rating"

    if restaurant['Votes'] > 1000:
        explanation += " (popular choice)"

    return explanation
```

Creates human-readable explanations for why a restaurant was recommended, increasing transparency and user trust

4-user interface

```
def setup_sidebar():
    """Setup the sidebar with user input controls"""
    st.sidebar.header("Restaurant Preferences")

    # Load data and initialize recommender
    df = preprocess_data()
    recommender = RestaurantRecommender(df)

    # Get unique values for filters
    cuisines = sorted(df['Primary Cuisine'].unique())
    locations = sorted(df['Location'].unique())
    price_buckets = ['Low', 'Medium', 'High']

    # User inputs
    selected_cuisine = st.sidebar.selectbox(
        "Preferred Cuisine",
        ['Any'] + cuisines
    )
```

```

    selected_location = st.sidebar.selectbox(
        "Preferred Location",
        ['Any'] + locations
    )

    selected_price = st.sidebar.selectbox(
        "Price Range",
        ['Any'] + price_buckets
    )

    top_n = st.sidebar.slider(
        "Number of Recommendations",
        min_value=1,
        max_value=10,
        value=5
    )
```

```
return {
    'cuisine': None if selected_cuisine == 'Any' else selected_cuisine,
    'location': None if selected_location == 'Any' else selected_location,
    'price_bucket': None if selected_price == 'Any' else selected_price,
    'top_n': top_n,
    'recommender': recommender
}
```

Creates an interactive sidebar with dropdown menus and sliders for:

- Cuisine selection (with "Any" option)
- Location selection (with "Any" option)
- Price range selection (with "Any" option)
- Number of recommendations to display

```
def display_recommendations(recommendations, recommender):
    """Display recommendations in the main area"""
    st.header("Recommended Restaurants")

    if recommendations.empty:
        st.warning("No restaurants match your criteria. Try broadening your search.")
        return

    # Create a map centered on the first recommendation
    map_center = [recommendations.iloc[0]['Latitude'], recommendations.iloc[0]['Longitude']]
    m = folium.Map(location=map_center, zoom_start=13)

    for _, row in recommendations.iterrows():
        # Add marker to map
        folium.Marker(
            [row['Latitude'], row['Longitude']],
            popup=row['Restaurant Name'],
            tooltip=f"{row['Restaurant Name']} - Rating: {row['Aggregate rating']}"
        ).add_to(m)
```

```
        # Create an expandable card for each recommendation
        with st.expander(f"🍽️ {row['Restaurant Name']} - ⭐ {row['Aggregate rating']}"):
            col1, col2 = st.columns([2, 1])

            with col1:
                st.subheader(row['Restaurant Name'])
                st.caption(f"📍 {row['Location']}")
                st.write(f"**Cuisine:** {row['Primary Cuisine'].title()}")
                st.write(f"**Price Range:** {row['Price Bucket']} (${row['Average Cost for tw
o']} for two)")
                st.write(f"**Rating:** {row['Aggregate rating']} from {row['Votes']} votes")
                st.write(f"**Why we recommend this:** {recommender.generate_explanation(row)}")

            with col2:
                st.metric("Average Cost for Two", f"${row['Average Cost for two']}")

    # Display the map
    st.header("Restaurant Locations")
    folium_static(m)
```

Displays recommendations in an interactive format including:

- Expandable restaurant cards with detailed information
- Interactive map showing restaurant locations
- Clear presentation of key information (rating, price, cuisine)
- Explanations for why each restaurant was recommended

```python
def main_ui():
    """Main user interface function"""
    st.set_page_config(page_title="Restaurant Recommender", page_icon="🍽️", layout="wide")
    st.title("🍽️ Knowledge-Based Restaurant Recommender")

    # Setup sidebar and get user preferences
    inputs = setup_sidebar()

    # Get recommendations
    recommendations = inputs['recommender'].recommend(
        cuisine=inputs['cuisine'],
        location=inputs['location'],
        price_bucket=inputs['price_bucket'],
        top_n=inputs['top_n']
    )

    # Display recommendations
    display_recommendations(recommendations, inputs['recommender'])
```

The main UI function that:

1. Configures the page settings
2. Sets up the sidebar controls
3. Gets recommendations based on user input
4. Displays the recommendations

5. Evaluation Module

Evaluates the recommendation system using predefined test cases and summarizes the results.

```python
def evaluate_recommendations(test_cases):
    """Evaluate the recommendation system with test cases"""
    df = preprocess_data()
    recommender = RestaurantRecommender(df)

    results = []

    for case in test_cases:
        recs = recommender.recommend(
            cuisine=case.get('cuisine'),
            location=case.get('location'),
            price_bucket=case.get('price_bucket'),
            top_n=case.get('top_n', 5)
        )

        results.append({
            'test_case': case,
            'num_recommendations': len(recs),
            'average_rating': recs['Aggregate rating'].mean() if not recs.empty else 0,
            'min_rating': recs['Aggregate rating'].min() if not recs.empty else 0,
            'max_rating': recs['Aggregate rating'].max() if not recs.empty else 0
        })
```

Evaluates the recommendation system by:

1. Running test cases through the recommender
2. Recording the number of recommendations and rating statistics
3. Returning a DataFrame with evaluation results

```python
def generate_test_cases():
    """Generate test cases for evaluation"""
    return [
        {'cuisine': 'indian', 'price_bucket': 'Medium', 'top_n': 3},
        {'cuisine': 'italian', 'location': 'Downtown', 'top_n': 5},
        {'price_bucket': 'High', 'top_n': 2},
        {'cuisine': 'japanese', 'location': 'Uptown', 'price_bucket': 'High', 'top_n': 4},
        {'cuisine': 'nonexistent', 'top_n': 3}  # Should return empty
    ]
```

Creates a variety of test cases to evaluate different aspects of the recommender:

- Cuisine filtering
- Location filtering
- Price filtering
- Edge cases (non-existent cuisine)

```python
def user_satisfaction_survey():
    """Simulate user satisfaction survey"""
    # In a real implementation, this would collect actual user feedback
    return {
        'satisfaction_score': 4.2,  # Average on Likert scale 1-5
        'relevance_score': 4.0,     # Average rating of recommendation relevance
        'usability_score': 4.5      # Average rating of interface usability
    }
```

Simulates user feedback (in a real system, this would collect actual user ratings).

```python
def generate_evaluation_report():
    """Generate comprehensive evaluation report"""
    test_cases = generate_test_cases()
    evaluation_results = evaluate_recommendations(test_cases)
    user_feedback = user_satisfaction_survey()

    report = {
        'test_case_results': evaluation_results,
        'user_feedback': user_feedback,
        'summary': {
            'success_rate': len(evaluation_results[evaluation_results['num_recommendations'] > 0]) / len(evaluation_results),
            'average_rating_across_tests': evaluation_results['average_rating'].mean(),
            'user_satisfaction': user_feedback['satisfaction_score']
        }
    }

    return report
```

Generates a comprehensive evaluation report including:

- Test case results
- User feedback
- Summary metrics (success rate, average rating, satisfaction score)

Main Execution:

```python
if __name__ == "__main__":
    report = generate_evaluation_report()
    print("Evaluation Report:")
    print("\nTest Case Results:")
    print(report['test_case_results'])
    print("\nUser Feedback:")
    print(report['user_feedback'])
    print("\nSummary:")
    print(report['summary'])
```

When run directly, the evaluation module generates and prints a detailed report of the system's performance.

conclusion:

This restaurant recommendation system demonstrates several best practices:

1. Modular Design: Separates concerns into distinct modules (data, engine, UI, evaluation)
2. Robustness: Handles missing data by generating synthetic data
3. Transparency: Provides explanations for recommendations
4. Evaluation: Includes comprehensive testing and evaluation
5. User Experience: Offers an interactive, visually appealing interface