Dataset Cleaning and Analysis

22/03/2025

stratified_history.csv
8.1 MB

Analyze the provided dataset to identify its key characteristics for data-driven decision-making. Upon receiving a specific analytical question, address it strictly using the dataset. Your deliverables must include a clear, concise answer to the question and the Python code that extracts and processes the data used to derive your answer. The dataset should be cleaned by filtering admission_period to include only values between 2006.1 and 2019.2, removing entries where enrollment_type is labeled as "Exempted", and excluding those where situation is "In Progress". The resulting cleaned dataset should be stored in a new DataFrame named df_filtered.

qwen-max-latest

Here is the analysis and Python code to clean the dataset as per the requirements:

Key Characteristics of the Dataset:

1. **Columns** : The dataset contains several columns related to student records, including `admission_period`, `enrollment_type`, `situation`, demographic information, academic performance metrics, and course details.
2. **Timeframe** : The `admission_period` column indicates when a student was admitted, spanning multiple years and semesters.
3. **Enrollment Types** : Includes categories like "Regular", "Canceled Failed All Due to Absences", "Graduate", etc.
4. **Situations** : Indicates the current status of the student, such as "Dropout", "Graduate", "In Progress", etc.
5. **Demographics** : Contains gender, race, age, and other personal details.
6. **Academic Data** : Includes grades, course names, academic units, and credit hours.

Cleaning Steps:

1. **Filter** `admission_period` : Include only values between 2006.1 and 2019.2.
2. **Remove "Exempted" Entries** : Exclude rows where `enrollment_type` is labeled as "Exempted".
3. **Exclude "In Progress" Situations** : Remove rows where `situation` is "In Progress".

Python Code:

Below is the Python code to clean the dataset and store the cleaned data in `df_filtered`.

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('stratified_history.csv', header=None)

# Assign column names based on dataset structure (adjust column names as needed)
columns = [
    'student_id', 'admission_year', 'admission_period', 'entrance_exam_score', 'admission_type',
    'course_code', 'enrollment_cancellation_reason', 'cancellation_period', 'semesters_enrolled'
    'situation', 'quota_reservation', 'gender', 'race', 'age_at_admission', 'max_age',
    'school_type', 'high_school_score', 'enrollment_type', 'grade', 'course_name',
    'credits', 'academic_unit', 'semester'
]
df.columns = columns

# Filter admission_period to include only values between 2006.1 and 2019.2
df_filtered = df[(df['admission_period'] >= 2006.1) & (df['admission_period'] <= 2019.2)]

# Remove entries where enrollment_type is labeled as "Exempted"
df_filtered = df_filtered[df_filtered['enrollment_type'] != 'Exempted']

# Exclude rows where situation is "In Progress"
df_filtered = df_filtered[df_filtered['situation'] != 'In Progress']

# Display the cleaned DataFrame
print(df_filtered.head())

# Save the cleaned dataset to a new CSV file (optional)
df_filtered.to_csv('cleaned_dataset.csv', index=False)
```

Explanation of the Code:

1. **Loading the Dataset** : The dataset is loaded using `pd.read_csv`. Column names are assigned based on the dataset's structure.
2. **Filtering** `admission_period` : Rows are filtered to include only those where `admission_period` falls within the range [2006.1, 2019.2].
3. **Removing "Exempted" Entries** : Rows with `enrollment_type` equal to "Exempted" are excluded.
4. **Excluding "In Progress" Situations** : Rows where `situation` is "In Progress" are removed.
5. **Saving the Cleaned Dataset** : The cleaned dataset is saved to a new CSV file for further use.

Deliverables:

1. **Cleaned DataFrame** : The resulting cleaned dataset is stored in `df_filtered`.
2. **Python Code** : The provided code performs all necessary cleaning steps.

If you have a specific analytical question, please provide it, and I will address it using the cleaned dataset.

Using historical graduation data, apply time-series models (ARIMA, Holt-Winters, LSTMs) to predict next year's graduation rate. Justify the best model based on predictive accuracy metrics (RMSE, MAPE, trend consistency).

qwen-max-latest

To address the question of predicting next year's graduation rate using historical data, we will follow a structured approach. This includes data preprocessing, model implementation, evaluation, and selection of the best model based on predictive accuracy metrics such as RMSE (Root Mean Squared Error), MAPE (Mean Absolute Percentage Error), and trend consistency.

Step 1: Data Preprocessing

The dataset contains historical graduation data, including admission periods, situations (`situation` column), and other relevant attributes. To prepare the data for time-series modeling:

1. **Filter Graduation Data** :
   - Extract rows where `situation` is "Graduate".
   - Aggregate the number of graduates by `admission_period` .
2. **Create Time-Series Data** :
   - Convert the aggregated graduation counts into a time-series format, with `admission_period` as the index.
3. **Check for Missing Values** :
   - Ensure there are no missing values in the time-series data. If gaps exist, use interpolation to fill them.
4. **Split Data** :
   - Divide the data into training and testing sets. For example, use data up to the last available year for training and reserve the most recent year(s) for testing.

Step 2: Model Implementation

We will implement three models: ARIMA, Holt-Winters, and LSTM. Each model will be trained on the training set and evaluated on the test set.

(a) ARIMA (AutoRegressive Integrated Moving Average)

ARIMA is a classical statistical model suitable for capturing trends and seasonality in time-series data.

- **Model Parameters** : Determine the optimal `(p, d, q)` parameters using ACF/PACF plots or grid search.
- **Implementation** :

```python
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Fit ARIMA model
arima_model = ARIMA(train_data, order=(p, d, q))
arima_fit = arima_model.fit()

# Predictions
arima_predictions = arima_fit.forecast(steps=len(test_data))

# Evaluate
arima_rmse = mean_squared_error(test_data, arima_predictions, squared=False)
arima_mape = mean_absolute_percentage_error(test_data, arima_predictions)
```

(b) Holt-Winters Exponential Smoothing

Holt-Winters is effective for data with both trend and seasonality.

- **Implementation** :

```python
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Fit Holt-Winters model
hw_model = ExponentialSmoothing(train_data, seasonal='add', seasonal_periods=4)
hw_fit = hw_model.fit()

# Predictions
hw_predictions = hw_fit.forecast(steps=len(test_data))

# Evaluate
hw_rmse = mean_squared_error(test_data, hw_predictions, squared=False)
hw_mape = mean_absolute_percentage_error(test_data, hw_predictions)
```

(c) LSTM (Long Short-Term Memory)

LSTM is a deep learning model capable of capturing complex patterns in sequential data.

- **Preprocessing for LSTM** :
  - Normalize the data using MinMaxScaler.
  - Create sequences of input-output pairs for training.
- **Implementation** :

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# Normalize data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(train_data.values.reshape(-1, 1))

# Create sequences
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)

seq_length = 10
X_train, y_train = create_sequences(scaled_data, seq_length)

# Build LSTM model
lstm_model = Sequential([
    LSTM(50, activation='relu', input_shape=(seq_length, 1)),
    Dense(1)
])
lstm_model.compile(optimizer='adam', loss='mse')

# Train model
lstm_model.fit(X_train, y_train, epochs=50, batch_size=32)

# Predictions
test_scaled = scaler.transform(test_data.values.reshape(-1, 1))
X_test, _ = create_sequences(test_scaled, seq_length)
lstm_predictions = lstm_model.predict(X_test)
lstm_predictions = scaler.inverse_transform(lstm_predictions)

# Evaluate
lstm_rmse = mean_squared_error(test_data[seq_length:], lstm_predictions, squared=False)
lstm_mape = mean_absolute_percentage_error(test_data[seq_length:], lstm_predictions)
```

Step 3: Model Evaluation

Evaluate each model using the following metrics:

1. **RMSE (Root Mean Squared Error)** :

- Measures the average magnitude of prediction errors.
- Lower RMSE indicates better performance.

2. **MAPE (Mean Absolute Percentage Error)** :
   - Measures the percentage error relative to actual values.
   - Useful for understanding the model's accuracy in percentage terms.
3. **Trend Consistency** :
   - Compare the predicted trend (increasing/decreasing) with the actual trend in the test data.

Step 4: Select the Best Model

Based on the evaluation metrics, select the model with the lowest RMSE and MAPE while maintaining trend consistency.

Python Code

Below is the complete Python code for implementing and evaluating the models:

python

```python
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Load and preprocess data
df = pd.read_csv('stratified_history.csv')
graduates = df[df['situation'] == 'Graduate'].groupby('admission_period').size()
graduates.index = pd.to_datetime(graduates.index.astype(str), format='%Y.%m')
train_data = graduates[:-1]  # Training set
test_data = graduates[-1:]  # Test set

# ARIMA
arima_model = ARIMA(train_data, order=(1, 1, 1))
arima_fit = arima_model.fit()
arima_predictions = arima_fit.forecast(steps=len(test_data))
arima_rmse = mean_squared_error(test_data, arima_predictions, squared=False)
arima_mape = mean_absolute_percentage_error(test_data, arima_predictions)

# Holt-Winters
hw_model = ExponentialSmoothing(train_data, seasonal='add', seasonal_periods=4)
hw_fit = hw_model.fit()
hw_predictions = hw_fit.forecast(steps=len(test_data))
hw_rmse = mean_squared_error(test_data, hw_predictions, squared=False)
hw_mape = mean_absolute_percentage_error(test_data, hw_predictions)

# LSTM
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(train_data.values.reshape(-1, 1))

def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)

seq_length = 10
X_train, y_train = create_sequences(scaled_data, seq_length)

lstm_model = Sequential([
    LSTM(50, activation='relu', input_shape=(seq_length, 1)),
    Dense(1)
])
lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.fit(X_train, y_train, epochs=50, batch_size=32)

test_scaled = scaler.transform(test_data.values.reshape(-1, 1))
X_test, _ = create_sequences(test_scaled, seq_length)
lstm_predictions = lstm_model.predict(X_test)
lstm_predictions = scaler.inverse_transform(lstm_predictions)

lstm_rmse = mean_squared_error(test_data[seq_length:], lstm_predictions, squared=False)
lstm_mape = mean_absolute_percentage_error(test_data[seq_length:], lstm_predictions)

# Results
print(f"ARIMA RMSE: {arima_rmse}, MAPE: {arima_mape}")
print(f"Holt-Winters RMSE: {hw_rmse}, MAPE: {hw_mape}")
print(f"LSTM RMSE: {lstm_rmse}, MAPE: {lstm_mape}")
```

Conclusion

After evaluating the models, the one with the lowest RMSE and MAPE, along with consistent trend predictions, will be selected as the best model. Based on typical performance:

- **ARIMA** : Good for linear trends and seasonality.
- **Holt-Winters** : Effective for data with strong seasonality.
- **LSTM** : Suitable for capturing complex, non-linear patterns.

For this dataset, if the graduation rates exhibit clear seasonality and linear trends, **Holt-Winters** may perform best. Otherwise, **LSTM** could outperform for more complex patterns.

Final Answer : The best model will depend on the evaluation metrics, but **Holt-Winters** or **LSTM** is likely to provide the most accurate predictions.