VAR-Based Forecasting Model

Overview

This model uses the **Vector AutoRegression (VAR)** technique to forecast future wait times and operational metrics for various attractions. The dataset contains historical wait times, capacity, and guest count information, which is used to generate predictions for the next two months.

Dependencies

To run this model, ensure you have the following Python packages installed:

```
pip install pandas numpy statsmodels
```

Data Preparation

1. Load Data

The dataset is loaded from a CSV file:

```
import pandas as pd

df = pd.read_csv(r"path_to_csv_file") #replace the path with path to
file
df.fillna(0, inplace=True)
```

2. Pivot the Data

The dataset is reshaped to create a time series format where each attraction's wait times, capacity, and guest count become columns:

```
df_pivot = df[['DEB_TIME', 'ENTITY_DESCRIPTION_SHORT','WAIT_TIME_MAX',
'NB_UNITS', 'CAPACITY', 'GUEST_CARRIED']]
df_pivot = df_pivot.pivot_table(index='DEB_TIME',
                                columns='ENTITY_DESCRIPTION_SHORT',
                                values=['WAIT_TIME_MAX', 'NB_UNITS',
'CAPACITY', 'GUEST_CARRIED'],
                                aggfunc='first')
df_pivot.columns = [f'{col[1]}<{col[0]}>' for col in df_pivot.columns]
df_pivot.reset_index(inplace=True)
```

3. Date Processing

Ensure timestamps are in the correct format and rounded to the nearest hour:

```
df_pivot['DEB_TIME'] =
pd.to_datetime(df_pivot['DEB_TIME']).dt.floor('H')
df_pivot.set_index('DEB_TIME', inplace=True)
```

4. Feature Engineering

Additional time-based features are added:

```
df_pivot['day_of_week'] = df_pivot.index.dayofweek.astype(float)
df_pivot['month'] = df_pivot.index.month.astype(float)
```

5. Handle Missing Values

Forward and backward fill missing values to avoid nulls in modeling:

```
df_pivot.fillna(method='ffill', inplace=True)
df_pivot.fillna(method='bfill', inplace=True)
```

Training the VAR Model

1. Select Optimal Lag

```
from statsmodels.tsa.api import VAR

model = VAR(df_pivot)
lag_selection = model.select_order(maxlags=60)  # Test up to 60 lags
optimal_lag = lag_selection.aic  # Use AIC criterion
print(f"Optimal Lag Order: {optimal_lag}")
```

2. Fit the Model

```
var_results = model.fit(optimal_lag)
```

Forecasting

1. Prepare Input for Forecast

Extract the last known values from the dataset to serve as input for predictions:

```
lag_input = df_pivot.values[-optimal_lag:]
```

2. Generate Forecast Timestamps

```
forecast_dates = pd.date_range(
    start=df_pivot.index[-1] + pd.Timedelta(hours=1),  # Start from
next hour
    periods=7*14,  # 60 days * 14 time slots per day (9:00 - 22:00)
    freq="H"
)
forecast_dates = forecast_dates[forecast_dates.hour.isin(range(9,
23))]  # Filter time slots
```

3. Run Forecasting

```
var_forecast = var_results.forecast(lag_input,
steps=len(forecast_dates))
```

4. Format Results

```
var_forecast_df = pd.DataFrame(var_forecast, index=forecast_dates,
columns=df_pivot.columns)
var_forecast_df = var_forecast_df.clip(lower=0)  # Ensure non-
negative values
```

5. Save Predictions

```
var_forecast_df.to_csv(r"C:/Users/PRASHANT/OneDrive/Desktop/HACKATHON
/var_forecast_df.csv")
```

Future Improvements

- **Fine-tune the model:** Try different lag selection criteria (BIC, HQIC, etc.).
- **Incorporate external factors:** Weather, holidays, and event schedules can further improve accuracy.
- **Use different forecasting methods:** Experiment with **Prophet** for individual time-series predictions.

Summary:

This README provides an **overview of the VAR-based time-series forecasting model**, including **data processing, model training, and forecasting steps**. The output is stored as `var_forecast_df.csv` and can be used for further analysis.