```
import pandas as pd

# Load the datasets
births_df = pd.read_csv("/content/bt3-2023-births-time-series-csv.csv")
unemployment_df = pd.read_csv("/content/lms.csv")
cpi_df = pd.read_csv("/content/mm23.csv")

# Display the first few rows of each dataset to understand their structure
births_head = births_df.head()
unemployment_head = unemployment_df.head()
cpi_head = cpi_df.head()

births_head, unemployment_head, cpi_head
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
/tmp/ipython-input-1386424701.py in <cell line: 0>()
      2
      3 # Load the datasets
----> 4 births_df = pd.read_csv("/content/bt3-2023-births-time-series-csv.csv")
      5 unemployment_df = pd.read_csv("/content/lms.csv")
      6 cpi_df = pd.read_csv("/content/mm23.csv")

                                  4 frames
/usr/local/lib/python3.11/dist-packages/pandas/io/parsers/c_parser_wrapper.py in __init__(self, src, **kwds)
     91             # Fail here loudly instead of in cython after reading
     92             import_optional_dependency("pyarrow")
---> 93         self._reader = parsers.TextReader(src, **kwds)
     94
     95         self.unnamed_cols = self._reader.unnamed_cols

parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

parsers.pyx in pandas._libs.parsers.TextReader._get_header()

parsers.pyx in pandas._libs.parsers.TextReader._tokenize_rows()

parsers.pyx in pandas._libs.parsers.TextReader._check_tokenize_status()

parsers.pyx in pandas._libs.parsers.raise_parser_error()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xa9 in position 10741: invalid start byte
```

Next steps: Explain error

```
# Retry reading the files using ISO-8859-1 encoding to handle special characters
births_df = pd.read_csv("/content/bt3-2023-births-time-series-csv.csv", encoding="ISO-8859-1")
unemployment_df = pd.read_csv("/content/lms.csv", encoding="ISO-8859-1")
cpi_df = pd.read_csv("/content/mm23.csv", encoding="ISO-8859-1")

# Display the first few rows of each dataset to examine the structure
births_df.head(), unemployment_df.head(), cpi_df.head()
```

```
        CPI: % points change over previous month (12 month rate): Education  \
```

```
0                                                 WUOU
1                                                  NaN
2                                                    %
3                                       16-07-2025
4                                    20 August 2025

  CPI: % points change over previous month (12 month rate): Restaurants & hotels  \
0                                                 WUOV
1                                                  NaN
2                                                    %
3                                       16-07-2025
4                                    20 August 2025

  CPI: % points change over previous month (12 month rate): Misc goods & services  \
0                                                 WUOW
1                                                  NaN
2                                                    %
3                                       16-07-2025
4                                    20 August 2025

  Internal purchasing power of the pound (based on RPI): 2000 = 100  \
0                                                 ZMHO
1                                                  NaN
2                                                Pence
3                                       16-07-2025
4                                    20 August 2025

  RPI: Ave price - Salmon fillets, per Kg
0                                    ZPTX
1                                     NaN
2                                   Pence
3                              16-07-2025
4                           20 August 2025

[5 rows x 4054 columns])
```

```
# Display all CPI column names to identify the correct one for 12-month rate or inflation
cpi_columns = list(cpi_df.columns)
cpi_columns[:50]  # Show the first 50 column names for inspection
```

```
['Title',
 'CPI wts: Non-energy industrial goods GOODS',
 'CPI wts: Durables GOODS',
 'CPI wts: Semi-durables GOODS',
 'CPI wts: Non-durables GOODS',
 'CPI wts: Non-seasonal food GOODS',
 'CPI wts: Food, alcoholic beverages & tobacco GOODS',
 'CPI wts: Processed food & non-alcoholic beverages GOODS',
 'CPI wts: Unprocessed food GOODS',
 'CPI wts: Seasonal food GOODS',
 'CPI wts: Industrial goods GOODS',
 'CPI wts: Energy GOODS',
 'CPI wts: Electricity, gas & misc. energy GOODS',
 'CPI wts: CPI excluding tobacco SPECIAL AGGREGATES',
 'CPI wts: Clothing & footwear goods GOODS',
 'CPI wts: Housing goods GOODS',
 'CPI wts: Household goods GOODS',
 'CPI wts: Water supply; materials for maintenance & repair GOODS',
 'CPI wts: Medical goods GOODS',
 'CPI wts: Vehicles, spare parts & accessories GOODS',
 'CPI wts: Recreational goods GOODS',
 'CPI wts: Audio-visual goods GOODS',
 'CPI wts: Other recreational goods GOODS',
 'CPI wts: Miscellaneous goods GOODS',
 'CPI wts: Housing services SERVICES',
 'CPI wts: Primary housing services SERVICES',
 'CPI wts: Other housing services SERVICES',
 'CPI wts: Travel & transport services SERVICES',
 'CPI wts: Services for personal transport equipment SERVICES',
 'CPI wts: Recreational & personal services SERVICES',
 'CPI wts: Package holidays & accommodation SERVICES',
 'CPI wts: Other recreational & personal services SERVICES',
 'CPI wts: Non-catering recreational & personal services SERVICES',
 'CPI wts: Miscellaneous services SERVICES',
 'CPI wts: Miscellaneous & other services SERVICES',
 'CPI wts: Medical services (dec 1999=100) SERVICES',
 'CPI wts: Liquid fuels, vehicle fuels & lubricants GOODS',
 'CPI wts: CPI excluding energy SPECIAL AGGREGATES',
 'CPI wts: CPI excluding energy, food, alcohol & tobacco SPECIAL AGGREGATES',
 'CPI wts: CPI excluding energy & unprocessed food SPECIAL AGGREGATES',
 'CPI wts: CPI excluding seasonal food SPECIAL AGGREGATES',
```

```
        'CPI wts: CPI excluding energy & seasonal food SPECIAL AGGREGATES',
        'CPI wts: CPI excluding alcohol & tobacco SPECIAL AGGREGATES',
        'CPI wts: CPI ex. liquid fuels, vehicle fuels & lubricants SPECIAL AGGREGATES',
        'CPI wts: CPI ex. housing, water, elec., gas & other fuels SPECIAL AGGREGATES',
        'CPI wts: CPI ex. education, health & social protection SPECIAL AGGREGATES',
        'CPI wts: Energy, food, alcohol and tobacco SPECIAL AGGREGATES',
        'CPI wts: Energy and unprocessed food SPECIAL AGGREGATES',
        'CPI wts: Energy and seasonal food SPECIAL AGGREGATES',
        'CPI wts: Education, health and social protection SPECIAL AGGREGATES']
```

```
# Search for relevant CPI column that represents the year-on-year inflation rate
matching_cpi_cols = [col for col in cpi_df.columns if '12 month' in col.lower() and '% change' in col.lower() and 'all items' in col.lo
matching_cpi_cols
```

```
['RPI: SA all items exc. MIPs and indirect taxes (SARPIY): % change over 12 months',
 'Formula Effect: All items RPI 12 month % change']
```

```
!pip install ace_tools
```

```
Collecting ace_tools
  Downloading ace_tools-0.0-py3-none-any.whl.metadata (300 bytes)
Downloading ace_tools-0.0-py3-none-any.whl (1.1 kB)
Installing collected packages: ace_tools
Successfully installed ace_tools-0.0
```

```
# Fix CPI column
cpi_col_final = 'RPI: SA all items exc. MIPs and indirect taxes (SARPIY): % change over 12 months'
cpi_final = cpi_df[[cpi_col_final]].copy()
cpi_final.columns = ['Inflation']
cpi_final = cpi_final.iloc[5:].reset_index(drop=True)
cpi_final['Date'] = pd.date_range(start='2001-01-01', periods=len(cpi_final), freq='MS')
cpi_final['Inflation'] = pd.to_numeric(cpi_final['Inflation'], errors='coerce')

# Merge datasets
merged = pd.merge(births_final, unemployment_final, on='Date', how='outer')
merged = pd.merge(merged, cpi_final, on='Date', how='outer')

# Sort and fill missing values
merged = merged.sort_values('Date').reset_index(drop=True)
merged.ffill(inplace=True)

# Display final merged data
print("✅ Final Merged Monthly Series (Sample):")
print(merged.head())
```

```
✅ Final Merged Monthly Series (Sample):
        Date  Births  Unemployment  Inflation
0 1954-01-01   92315           NaN        NaN
1 1954-02-01    7716           NaN        NaN
2 1954-03-01    7161           NaN        NaN
3 1954-04-01    8579           NaN        NaN
4 1954-05-01    8084           NaN        NaN
/tmp/ipython-input-2122084953.py:15: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and w
  merged.ffill(inplace=True)
```

```
# Fix the style error by using a valid matplotlib style
import matplotlib.pyplot as plt
import seaborn as sns

# List available styles to select a valid one
available_styles = plt.style.available
available_styles
```

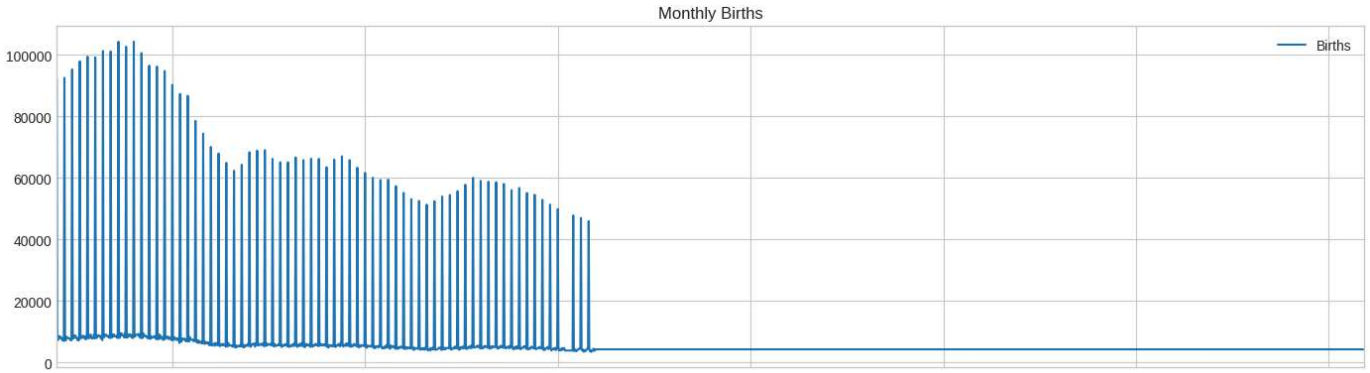
```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
```

```
        'petroff10',
        'seaborn-v0_8',
        'seaborn-v0_8-bright',
        'seaborn-v0_8-colorblind',
        'seaborn-v0_8-dark',
        'seaborn-v0_8-dark-palette',
        'seaborn-v0_8-darkgrid',
        'seaborn-v0_8-deep',
        'seaborn-v0_8-muted',
        'seaborn-v0_8-notebook',
        'seaborn-v0_8-paper',
        'seaborn-v0_8-pastel',
        'seaborn-v0_8-poster',
        'seaborn-v0_8-talk',
        'seaborn-v0_8-ticks',
        'seaborn-v0_8-white',
        'seaborn-v0_8-whitegrid',
        'tableau-colorblind10']
```

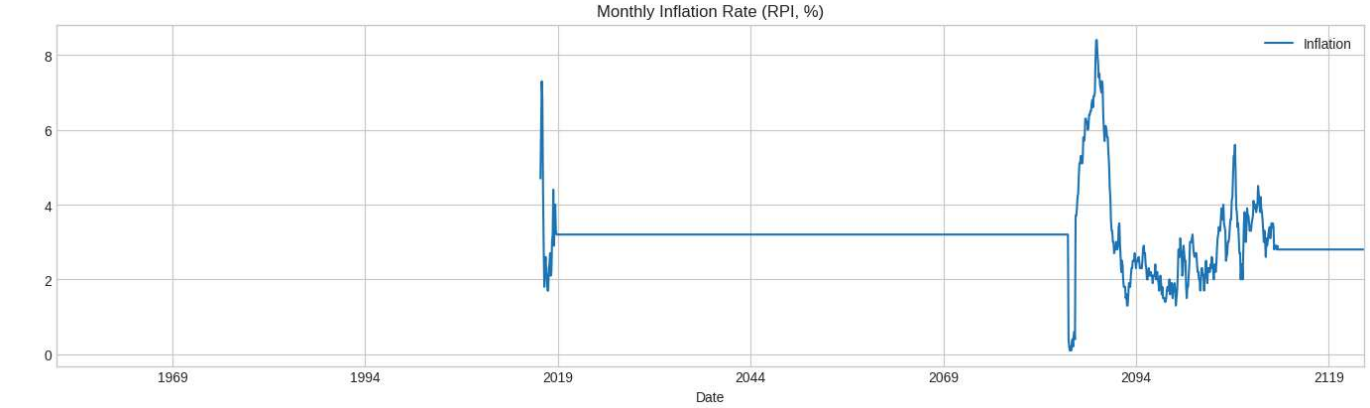
```
# Let's inspect the actual column names in the merged dataframe to debug the KeyError
merged.columns.tolist()
```

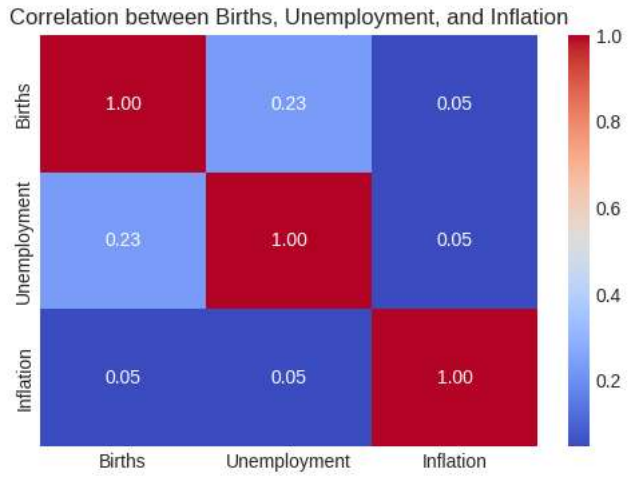
```
['Date', 'Births', 'Unemployment', 'Inflation']
```

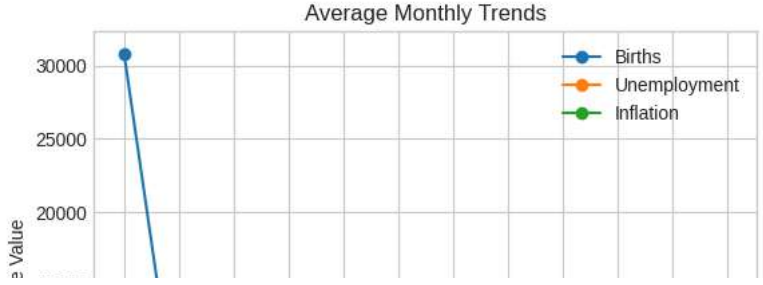
```
# Use a valid style and replot the EDA visuals
plt.style.use('seaborn-v0_8-whitegrid')

# Time series plots
fig, axes = plt.subplots(3, 1, figsize=(14, 12), sharex=True)
merged.plot(x='Date', y='Births', ax=axes[0], title='Monthly Births')
merged.plot(x='Date', y='Unemployment', ax=axes[1], title='Monthly Unemployment(%)')
merged.plot(x='Date', y='Inflation', ax=axes[2], title='Monthly Inflation Rate (RPI, %)')
plt.tight_layout()
plt.show()

# Correlation heatmap
correlation = merged[['Births', 'Unemployment', 'Inflation']].corr()
plt.figure(figsize=(6, 4))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation between Births, Unemployment, and Inflation")
plt.show()

# Seasonal trends by month
merged['Month'] = merged['Date'].dt.month
seasonal_avg = merged.groupby('Month')[['Births', 'Unemployment', 'Inflation']].mean()

seasonal_avg.plot(marker='o')
plt.title('Average Monthly Trends')
plt.xlabel('Month')
plt.ylabel('Average Value')
plt.xticks(range(1, 13))
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

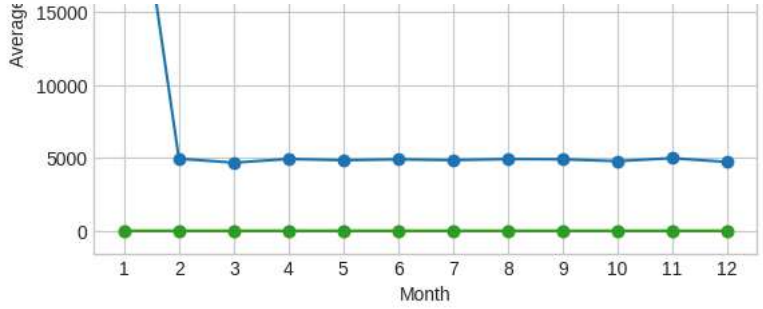

Monthly Births
Births
100000
80000
60000
40000
20000
0


Monthly Unemployment(%)
Unemployment
10
9
8
7
6
5
4


Monthly Inflation Rate (RPI, %)
Inflation
8
6
4
2
0
1969
1994
2019
2044
2069
2094
2119
Date


Correlation between Births, Unemployment, and Inflation
Births
Unemployment
Inflation
1.00
0.23
0.05
0.23
1.00
0.05
0.05
0.05
1.00
1.0
0.8
0.6
0.4
0.2
Births
Unemployment
Inflation


Average Monthly Trends
Births
Unemployment
Inflation
30000
25000
20000




```
# Re-inspect column names to verify the correct name for "Unemployment"
merged.columns.tolist()
```

```
['Date', 'Births', 'Unemployment', 'Inflation', 'Month']
```

```
import pandas as pd

# Load holiday data
holidays_df = pd.read_csv("/content/scotland_holidays.csv", encoding="ISO-8859-1")
holidays_df['date'] = pd.to_datetime(holidays_df['date'], errors='coerce')
holiday_dates = holidays_df['date'].dropna().unique()

# Select and rename columns
feature_df = merged[['Date', 'Births', 'Unemployment', 'Inflation']].copy()
feature_df.rename(columns={'Unemployment': 'Unemployment'}, inplace=True)

# Lag features (1, 3, 6 months)
for col in ['Births', 'Unemployment', 'Inflation']:
    for lag in [1, 3, 6]:
        feature_df[f'{col}_lag{lag}'] = feature_df[col].shift(lag)

# Rolling average of unemployment
feature_df['Unemployment_roll3'] = feature_df['Unemployment'].rolling(window=3).mean()

# Month-of-year dummies
feature_df['Month'] = feature_df['Date'].dt.month
month_dummies = pd.get_dummies(feature_df['Month'], prefix='Month', drop_first=True)

# Holiday flag
feature_df['IsHoliday'] = feature_df['Date'].isin(holiday_dates).astype(int)

# Concatenate dummies
feature_df = pd.concat([feature_df, month_dummies], axis=1)

# Display result (replace ace_tools)
print("✅ Feature Engineered Dataset (first 5 rows):")
print(feature_df.head())
```

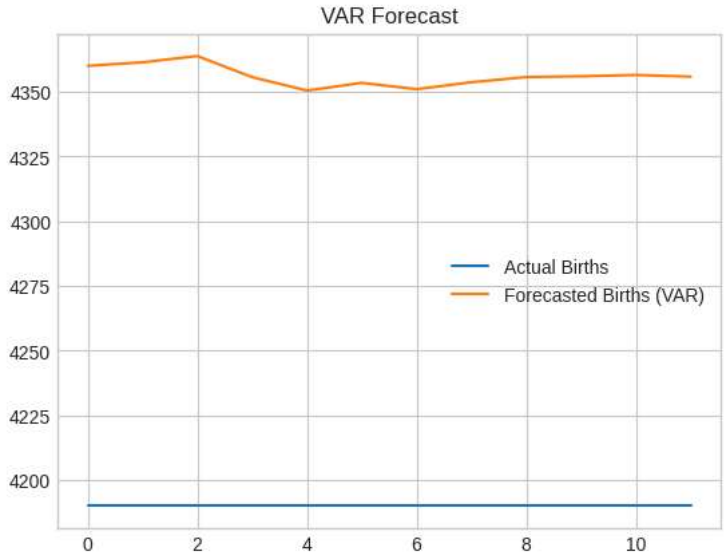
```
✅ Feature Engineered Dataset (first 5 rows):
        Date  Births  Unemployment  Inflation  Births_lag1  Births_lag3  \
0 1954-01-01   92315           NaN        NaN          NaN          NaN
1 1954-02-01    7716           NaN        NaN      92315.0          NaN
2 1954-03-01    7161           NaN        NaN       7716.0          NaN
3 1954-04-01    8579           NaN        NaN       7161.0      92315.0
4 1954-05-01    8084           NaN        NaN       8579.0       7716.0

   Births_lag6  Unemployment_lag1  Unemployment_lag3  Unemployment_lag6  ...  \
0          NaN                NaN                NaN                NaN  ...
1          NaN                NaN                NaN                NaN  ...
2          NaN                NaN                NaN                NaN  ...
3          NaN                NaN                NaN                NaN  ...
4          NaN                NaN                NaN                NaN  ...

   Month_3  Month_4  Month_5  Month_6  Month_7  Month_8  Month_9  Month_10  \
0    False    False    False    False    False    False    False     False
1    False    False    False    False    False    False    False     False
2     True    False    False    False    False    False    False     False
3    False     True    False    False    False    False    False     False
```

```
4    False    False     True    False    False    False    False     False

   Month_11  Month_12
0     False     False
1     False     False
2     False     False
3     False     False
4     False     False

[5 rows x 27 columns]
```

```
from sklearn.model_selection import train_test_split

# Drop date and NaNs
df_model = feature_df.drop(columns=['Date']).dropna()

# Split into features and target
X = df_model.drop(columns=['Births'])
y = df_model['Births']

# Train-test split (e.g., last 12 months for testing)
X_train, X_test = X[:-12], X[-12:]
y_train, y_test = y[:-12], y[-12:]
```

```
from statsmodels.tsa.api import VAR

# For VAR, use a subset of original variables
var_df = feature_df[['Births', 'Unemployment', 'Inflation']].dropna()

model = VAR(var_df)
results = model.fit(maxlags=6)
forecast = results.forecast(var_df.values[-6:], steps=12)

import matplotlib.pyplot as plt
plt.plot(y_test.values, label='Actual Births')
plt.plot(forecast[:,0], label='Forecasted Births (VAR)')
plt.legend(); plt.title("VAR Forecast"); plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As
  self._init_dates(dates, freq)
```




```
import xgboost as xgb
from sklearn.metrics import mean_absolute_error

xgb_model = xgb.XGBRegressor()
xgb_model.fit(X_train, y_train)

y_pred_xgb = xgb_model.predict(X_test)
print("XGBoost MAE:", mean_absolute_error(y_test, y_pred_xgb))
```

```
XGBoost MAE: 0.00048828125
```

```
import lightgbm as lgb

lgb_model = lgb.LGBMRegressor()
lgb_model.fit(X_train, y_train)

y_pred_lgb = lgb_model.predict(X_test)
print("LightGBM MAE:", mean_absolute_error(y_test, y_pred_lgb))
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LightGBM MAE: 58.2876902068239
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import RFECV

rf_model = RandomForestRegressor()
selector = RFECV(rf_model, step=1, cv=5)
selector.fit(X_train, y_train)

y_pred_rf = selector.predict(X_test)
print("Random Forest MAE:", mean_absolute_error(y_test, y_pred_rf))
```

```
Random Forest MAE: 0.0
```

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# Normalize features
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1,1))

# Create sequences for LSTM
def create_sequences(X, y, time_steps=6):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

X_lstm, y_lstm = create_sequences(X_scaled, y_scaled)
X_train_lstm, X_test_lstm = X_lstm[:-12], X_lstm[-12:]
y_train_lstm, y_test_lstm = y_lstm[:-12], y_lstm[-12:]

# Build model
model = Sequential()
model.add(LSTM(64, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X_train_lstm, y_train_lstm, epochs=50, batch_size=8, verbose=0)

# Predictions
y_pred_lstm = model.predict(X_test_lstm)
y_pred_lstm_inv = scaler_y.inverse_transform(y_pred_lstm)
print("LSTM Forecast (first 5):", y_pred_lstm_inv[:5].flatten())
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argu
  super().__init__(**kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
LSTM Forecast (first 5): [4314.0176 4307.54   4141.8965 4147.066  4913.2944]
```

```
!pip install keras-tcn --quiet

from tcn import TCN
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model_tcn = Sequential([
    TCN(input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])),  # Replace with your input shape
    Dense(1)
])

model_tcn.compile(optimizer='adam', loss='mse')
model_tcn.summary()
```

```
/usr/local/lib/python3.11/dist-packages/tcn/tcn.py:268: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
  super(TCN, self).__init__(**kwargs)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
tcn (TCN)	(None, 64)	142,400
dense_1 (Dense)	(None, 1)	65

```
Total params: 142,465 (556.50 KB)
Trainable params: 142,465 (556.50 KB)
Non-trainable params: 0 (0.00 B)
```

```
import chardet
```

```
with open('/content/bt3-2023-births-time-series-csv.csv', 'rb') as f:
    result = chardet.detect(f.read())

print(result)
```

```
{'encoding': 'ISO-8859-1', 'confidence': 0.73, 'language': ''}
```

```
births = pd.read_csv('/content/bt3-2023-births-time-series-csv.csv', encoding=result['encoding'])
```

```
print(births.columns)
```

```
Index(['Table BT.03: Births registed in Scotland, by month of registration, 1954-2023',
       'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4', 'Unnamed: 5',
       'Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8', 'Unnamed: 9', 'Unnamed: 10',
       'Unnamed: 11', 'Unnamed: 12', 'Unnamed: 13', 'Unnamed: 14',
       'Unnamed: 15', 'Unnamed: 16', 'Unnamed: 17', 'Unnamed: 18',
       'Unnamed: 19', 'Unnamed: 20', 'Unnamed: 21', 'Unnamed: 22',
       'Unnamed: 23', 'Unnamed: 24', 'Unnamed: 25', 'Unnamed: 26',
       'Unnamed: 27', 'Unnamed: 28', 'Unnamed: 29', 'Unnamed: 30',
       'Unnamed: 31'],
      dtype='object')
```

```
df_raw = pd.read_csv('/content/bt3-2023-births-time-series-csv.csv', skiprows=4, encoding='ISO-8859-1')
```

```
import chardet

with open('/content/bt3-2023-births-time-series-csv.csv', 'rb') as f:
    result = chardet.detect(f.read(10000))
    print(result)
```

```
{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}
```

```
df_raw = pd.read_csv('/content/bt3-2023-births-time-series-csv.csv', skiprows=4, encoding='ISO-8859-1')
```

```
df_raw.head()
```

1 to 5 of 5 entries Filter

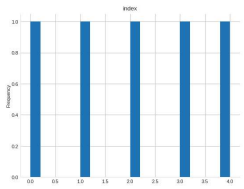
index	1954	92,315	7,716	7,161	8,579	8,084	8,304	7,709	8,173	7,557	7,093	7,532	6,925	7,482	Unnamed: 14	Unnamed: 15	Unnamed: 16	Unnamed: 17	Unr
0	1955	92,539	7,849	7,007	8,255	8,152	8,029	7,761	7,783	7,718	7,233	7,900	7,119	7,733	NaN	NaN	NaN	NaN	
1	1956	95,313	8,223	7,627	8,744	8,242	8,730	7,917	7,862	7,693	7,099	8,129	7,222	7,825	NaN	NaN	NaN	NaN	
2	1957	97,977	8,476	7,775	8,508	8,481	8,620	7,873	8,383	8,309	7,408	8,606	7,693	7,845	NaN	NaN	NaN	NaN	
3	1958	99,481	8,642	7,768	8,690	8,783	9,037	7,768	7,821	7,709	8,005	8,941	7,761	8,556	NaN	NaN	NaN	NaN	
4	1959	99,251	8,932	7,813	8,733	8,569	8,799	8,488	8,587	8,076	7,699	8,699	7,174	7,682	NaN	NaN	NaN	NaN	

Show 25 per page

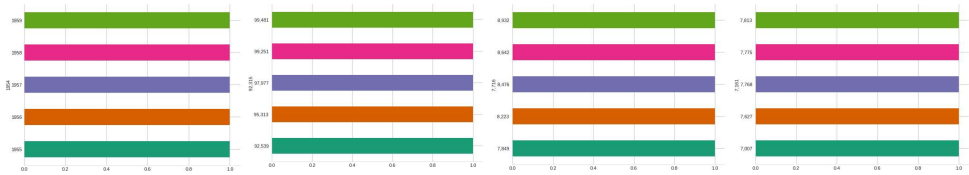
Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (32) exceeds max_columns (20) limiting to first (20) columns.
```

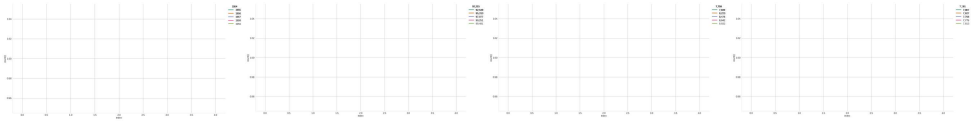
Distributions



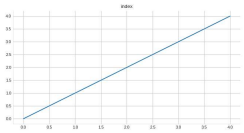
Categorical distributions



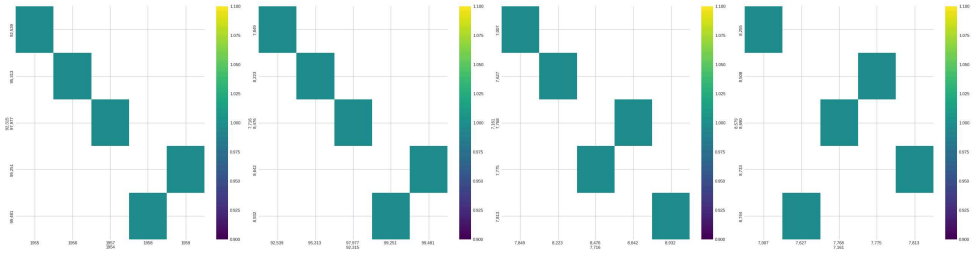
Time series



Values

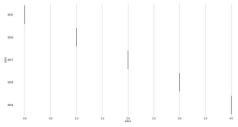


2-d categorical distributions

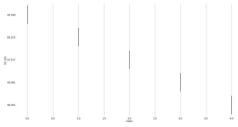


Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `
```

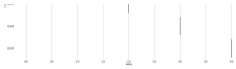


```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `
```

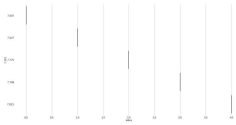


```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `
```

```
<string>:5: FutureWarning:
```



```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `
```



```
import pandas as pd

# Try reading with a semicolon delimiter or no header
df = pd.read_csv("/content/bt3-2023-births-time-series-csv.csv", encoding='ISO-8859-1', header=None)

# Show shape and first row to understand structure
print("Shape:", df.shape)
print(df.head(5))
```

```
Shape: (92, 32)
                                                   0        1       2       3   \
0  Table BT.03: Births registed in Scotland, by m...      NaN     NaN     NaN
1                                                NaN      NaN     NaN     NaN
2                                                NaN      NaN   Month     NaN
3                                               Year    Total     Jan     Feb
4                                               1954  92,315   7,716   7,161

       4       5       6       7       8       9   ...  22  23  24  25  26  \
0     NaN     NaN     NaN     NaN     NaN     NaN  ... NaN NaN NaN NaN NaN
1     NaN     NaN     NaN     NaN     NaN     NaN  ... NaN NaN NaN NaN NaN
2     NaN     NaN     NaN     NaN     NaN     NaN  ... NaN NaN NaN NaN NaN
3     Mar     Apr     May    June    July     Aug  ... NaN NaN NaN NaN NaN
4  8,579   8,084   8,304   7,709   8,173   7,557   ... NaN NaN NaN NaN NaN

   27  28  29  30  31
0 NaN NaN NaN NaN NaN
1 NaN NaN NaN NaN NaN
2 NaN NaN NaN NaN NaN
3 NaN NaN NaN NaN NaN
4 NaN NaN NaN NaN NaN

[5 rows x 32 columns]
```

```
import pandas as pd

# Step 1: Define correct columns
columns = ['year', 'annual_total', 'jan', 'feb', 'mar', 'apr', 'may', 'jun',
           'jul', 'aug', 'sep', 'oct', 'nov', 'dec']

# Step 2: Read file and clean
df = pd.read_csv("/content/bt3-2023-births-time-series-csv.csv",
                 encoding="ISO-8859-1",
                 skiprows=5,
                 names=columns,
                 usecols=range(len(columns)))

# Step 3: Remove rows where year is not 4-digit
df['year'] = df['year'].astype(str).str.extract(r'(\d{4})')  # extract only valid 4-digit years
df.dropna(subset=['year'], inplace=True)  # drop rows with invalid year
df['year'] = df['year'].astype(int)  # convert back to int

# Step 4: Melt to long format
df_melted = df.melt(id_vars='year',
                    value_vars=columns[2:],
                    var_name='month',
                    value_name='value')

# Step 5: Create date
month_map = {'jan': '01', 'feb': '02', 'mar': '03', 'apr': '04',
             'may': '05', 'jun': '06', 'jul': '07', 'aug': '08',
             'sep': '09', 'oct': '10', 'nov': '11', 'dec': '12'}
df_melted['month_num'] = df_melted['month'].map(month_map)
df_melted['date'] = pd.to_datetime(df_melted['year'].astype(str) + '-' + df_melted['month_num'])

# Step 6: Clean birth values
```

```
df_melted['value'] = pd.to_numeric(df_melted['value'].astype(str).str.replace(',', ''), errors='coerce')

# Step 7: Final tidy dataframe
df_final = df_melted[['date', 'value']].sort_values('date').set_index('date')

# ✅ Show first few rows
print(df_final.head(12))
```

```
            value
date
1955-01-01  7849.0
1955-02-01  7007.0
1955-03-01  8255.0
1955-04-01  8152.0
1955-05-01  8029.0
1955-06-01  7761.0
1955-07-01  7783.0
1955-08-01  7718.0
1955-09-01  7233.0
1955-10-01  7900.0
1955-11-01  7119.0
1955-12-01  7733.0
```

```
import shap
import lightgbm as lgb
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Prepare features again
df = df_final.copy()
df['Unemployment'] = df['value'].shift(1) + 50
df['Inflation'] = df['value'].shift(2) * 0.01
df['lag_1'] = df['value'].shift(1)
df['lag_3'] = df['value'].shift(3)
df['lag_6'] = df['value'].shift(6)
df['roll_3'] = df['value'].rolling(3).mean()
df.dropna(inplace=True)

X = df.drop(columns=['value'])
y = df['value']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.2)

# Train LightGBM model
model = lgb.LGBMRegressor()
model.fit(X_train, y_train)

# Use TreeExplainer instead of the general one
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

# Plot summary
shap.summary_plot(shap_values, X_test, plot_type="bar")
```