```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from pandas_datareader import wb

st.set_page_config(page_title="Latin America Regression Explorer", layout="wide")

# Wealthiest Latin American countries by GDP
latin_countries = {
    "Argentina": "ARG",
    "Brazil": "BRA",
    "Chile": "CHL",
    "Colombia": "COL",
    "Mexico": "MEX",
    "Peru": "PER",
    "Ecuador": "ECU",
    "Panama": "PAN",
    "Uruguay": "URY",
    "Dominican Republic": "DOM"
}

# World Bank indicators
indicators = {
    "Population": "SP.POP.TOTL",
    "Unemployment rate": "SL.UEM.TOTL.ZS",
    "Education levels (0-25)": "SE.SEC.CUAT.UP.ZS",  # proxy: completed upper secondary
    "Life expectancy": "SP.DYN.LE00.IN",
    "Average wealth": "NY.GNP.PCAP.CD",  # proxy: GNI per capita
    "Average income": "NY.ADJ.NNTY.PC.CD",  # proxy: Adjusted net nat'l income
    "Birth rate": "SP.DYN.CBRT.IN",
    "Immigration out of the country": "SM.EMI.TOTL",  # emigrants
    "Murder Rate": "VC.IHR.PSRC.P5"  # intentional homicide rate
}

st.title("📊 Latin America Regression Explorer")
st.write("Analyze historical socioeconomic trends of the wealthiest Latin American countries (last
70 years) using polynomial regression.")

# Sidebar controls
st.sidebar.header("Controls")
indicator = st.sidebar.selectbox("Select category", list(indicators.keys()))
```

```
selected_countries = st.sidebar.multiselect("Select countries", list(latin_countries.keys()),
default=["Brazil", "Mexico"])
degree = st.sidebar.slider("Polynomial degree", 3, 7, 3)
step = st.sidebar.slider("Year increment (x-axis step)", 1, 10, 1)
future_years = st.sidebar.slider("Extrapolate into the future (years)", 0, 50, 20)

# Fetch data
data = wb.download(indicator=indicators[indicator],
                   country=[latin_countries[c] for c in selected_countries],
                   start=1950, end=2023)

data = data.reset_index().pivot(index="year", columns="country",
values=indicators[indicator]).sort_index()
st.subheader("📑 Raw Data (Editable)")
edited_data = st.data_editor(data, num_rows="dynamic")

# Regression + plotting
fig, ax = plt.subplots(figsize=(10,6))
x_all = np.array(edited_data.index)
x_future = np.arange(x_all.min(), x_all.max()+future_years+1)

for country in edited_data.columns:
    y = edited_data[country].dropna()
    x = y.index.values.reshape(-1,1)
    if len(x) < degree+1:  # skip if not enough data
        continue

    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(x)
    model = LinearRegression().fit(X_poly, y.values)

    # Equation display
    coeffs = model.coef_
    intercept = model.intercept_
    equation_terms = [f"{coeff:.2e}x^{i}" for i, coeff in enumerate(coeffs) if i > 0]
    equation = f"{intercept:.2e} + " + " + ".join(equation_terms)
    st.write(f"**{country} model equation:** {equation}")

    # Prediction
    X_all_poly = poly.transform(x_future.reshape(-1,1))
    y_pred = model.predict(X_all_poly)

    # Plot scatter
    ax.scatter(x, y, label=f"{country} data")
```

```
    # Plot regression + extrapolation
    ax.plot(x_future, y_pred, label=f"{country} regression")

    if future_years > 0:
        ax.axvline(x_all.max(), color="gray", linestyle="--")
        ax.plot(x_future[x_future > x_all.max()],
                y_pred[x_future > x_all.max()],
                linestyle="--", color="red", label=f"{country} extrapolation")

ax.set_xlabel("Year")
ax.set_ylabel(indicator)
ax.set_title(f"{indicator} over time")
ax.legend()
st.pyplot(fig)

# Function analysis (for first selected country)
if selected_countries:
    country = selected_countries[0]
    y = edited_data[country].dropna()
    x = y.index.values.reshape(-1,1)
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(x)
    model = LinearRegression().fit(X_poly, y.values)

    coeffs = model.coef_
    intercept = model.intercept_
    p = np.poly1d(np.concatenate(([intercept], coeffs[::-1])))  # approx poly

    # Derivative
    dp = np.polyder(p)
    ddp = np.polyder(dp)

    years = np.linspace(x.min(), x.max(), 500)
    slopes = dp(years)
    accels = ddp(years)

    max_year = years[np.argmax(p(years))]
    min_year = years[np.argmin(p(years))]
    fastest_growth_year = years[np.argmax(slopes)]
    fastest_decline_year = years[np.argmin(slopes)]

    st.subheader("📈 Function Analysis")
    st.write(f"- Local maximum around {int(max_year)} with value {p(max_year):.2f}")
```

```
    st.write(f"- Local minimum around {int(min_year)} with value {p(min_year):.2f}")
    st.write(f"- Growing fastest around {int(fastest_growth_year)} at rate
{dp(fastest_growth_year):.2f} units/year")
    st.write(f"- Declining fastest around {int(fastest_decline_year)} at rate
{dp(fastest_decline_year):.2f} units/year")
    st.write(f"- Domain: {x.min()}–{x.max()} years, Range: {y.min():.2f}–{y.max():.2f}")

# Prediction tool
st.subheader("🔮 Prediction & Average Rate of Change")
year_input = st.number_input("Enter year for prediction", min_value=1950, max_value=2100,
value=2030)
if selected_countries:
    country = selected_countries[0]
    y = edited_data[country].dropna()
    x = y.index.values.reshape(-1,1)
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(x)
    model = LinearRegression().fit(X_poly, y.values)
    pred_val = model.predict(poly.transform([[year_input]]))[0]
    st.write(f"Prediction for {country} in {year_input}: **{pred_val:.2f}** {indicator} units")

# Average rate of change
col1, col2 = st.columns(2)
with col1:
    year1 = st.number_input("Year 1", min_value=1950, max_value=2100, value=2000)
with col2:
    year2 = st.number_input("Year 2", min_value=1950, max_value=2100, value=2010)

if year2 > year1 and selected_countries:
    val1 = model.predict(poly.transform([[year1]]))[0]
    val2 = model.predict(poly.transform([[year2]]))[0]
    avg_rate = (val2 - val1) / (year2 - year1)
    st.write(f"Average rate of change between {year1} and {year2}: {avg_rate:.2f} units/year")

# Printer friendly option
st.download_button("🖨️ Download printer-friendly CSV", edited_data.to_csv().encode("utf-8"),
"data.csv", "text/csv")
```