**Economic Trends Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **Hasan** | **21F-9212** | **BCS-5B** |
| **2** | **Qasim** | **22F-3100** | **BAI-3B** |
| **3** | **Maryam** | **22F-3168** | **BAI-3B** |
| **4** | **Usman** | **21F-9178** | **BCS-5B** |
| **5** | **Abdullah** | **21F-9169** | **BCS-5B** |

**Submitted to:**

**Dr. Haris Khurram**

**Probability and Statistics**

1. **Problem Statement:**

We want to figure out what might happen to the economies of different countries in the future. To do this, we're going to look at how their economies have been doing in the past, especially at things like GDP and data from the International Monetary Fund (IMF). Our goal is to use this information to make educated guesses about what could happen to their economies in the coming years.

1. **Objectives:**
   1. **Graphical and Tabular Representation:**

Utilize graphical and tabular methods to visually represent economic data, enhancing the accessibility and interpretability of trends and patterns.

* 1. **Descriptive Statistical Measures:**

Employ descriptive statistical measures to quantify and summarize key features of the economic data, providing a comprehensive overview of central tendencies and variability.

* 1. **Probability Methods/Distribution:**

Apply probability methods and distributions to model uncertainty and assess the likelihood of different economic outcomes, contributing to a nuanced understanding of future scenarios.

* 1. **Regression Modeling and Predictions:**

Implement regression modeling techniques to capture relationships between economic variables and make predictions about future GDP values for the selected countries.

* 1. **Confidence Interval of Descriptive Measures:**

Calculate confidence intervals for both descriptive measures and regression estimates, providing a measure of precision and uncertainty around key statistical findings.

1. **Data Description:**

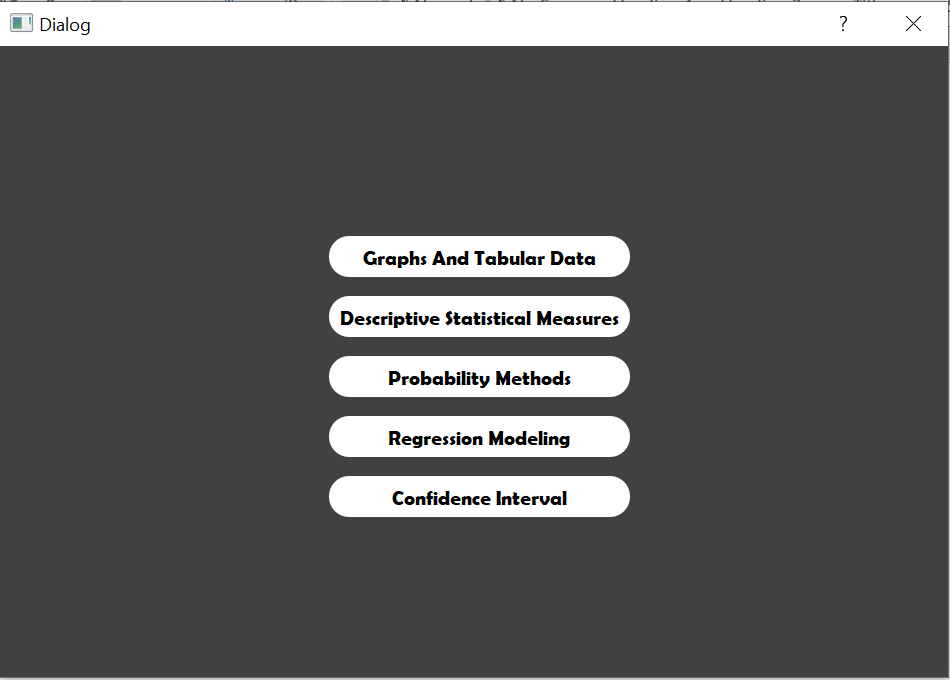
Link of data is:

<https://www.kaggle.com/code/scratchpad/notebook26ce71c4a9/edit>

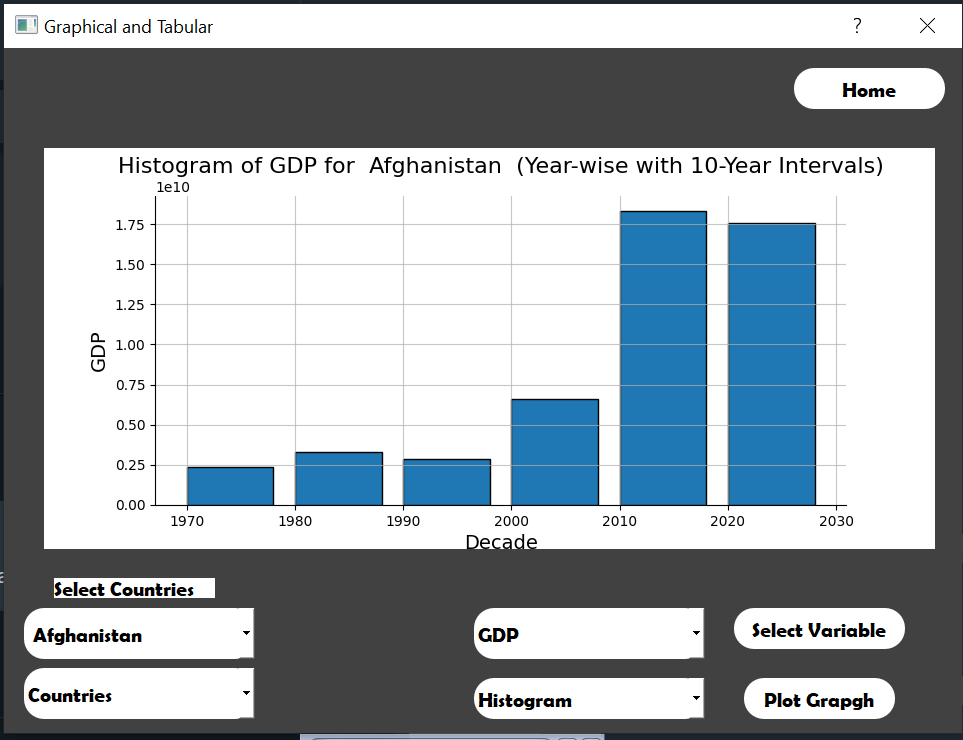
The dataset includes essential economic variables for various countries spanning the years **1970 to 2021**. **Country Name** serves as the categorical identifier, while **Year** provides a temporal context. **AMA** represents the Age-Adjusted Mortality Rate, a crucial health metric. **IMF Rate** indicates the exchange rate reported by the International Monetary Fund. **Population** quantifies the total residents, **GNI** reflects Gross National Income, and **GDP** represents Gross Domestic Product, offering comprehensive economic insights. Additionally, **Exports** and **Imports** signify the value of goods and services traded internationally.

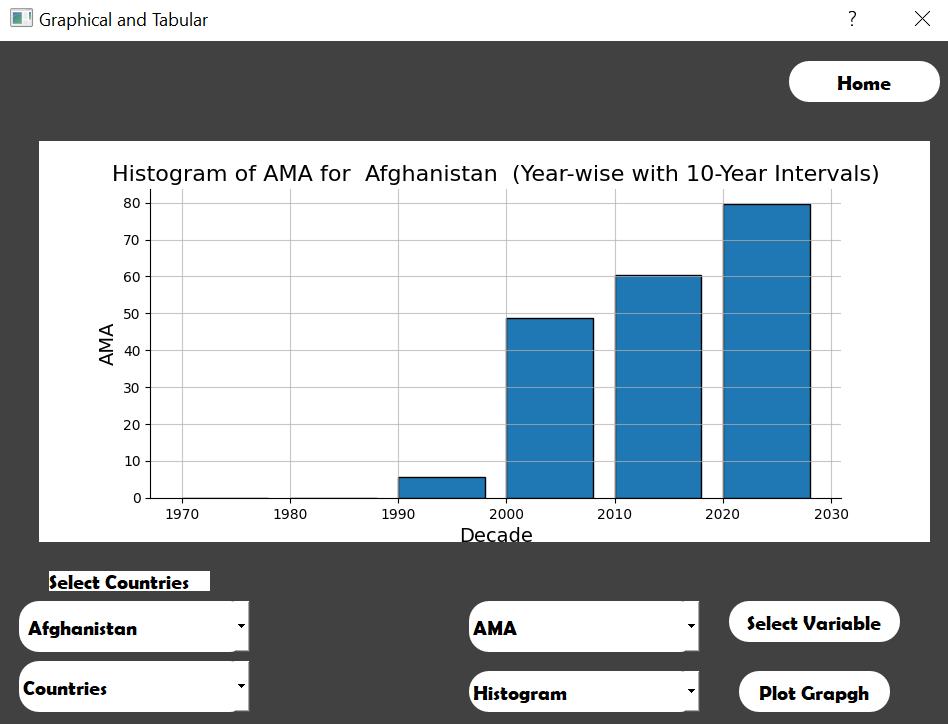
1. **Results:**

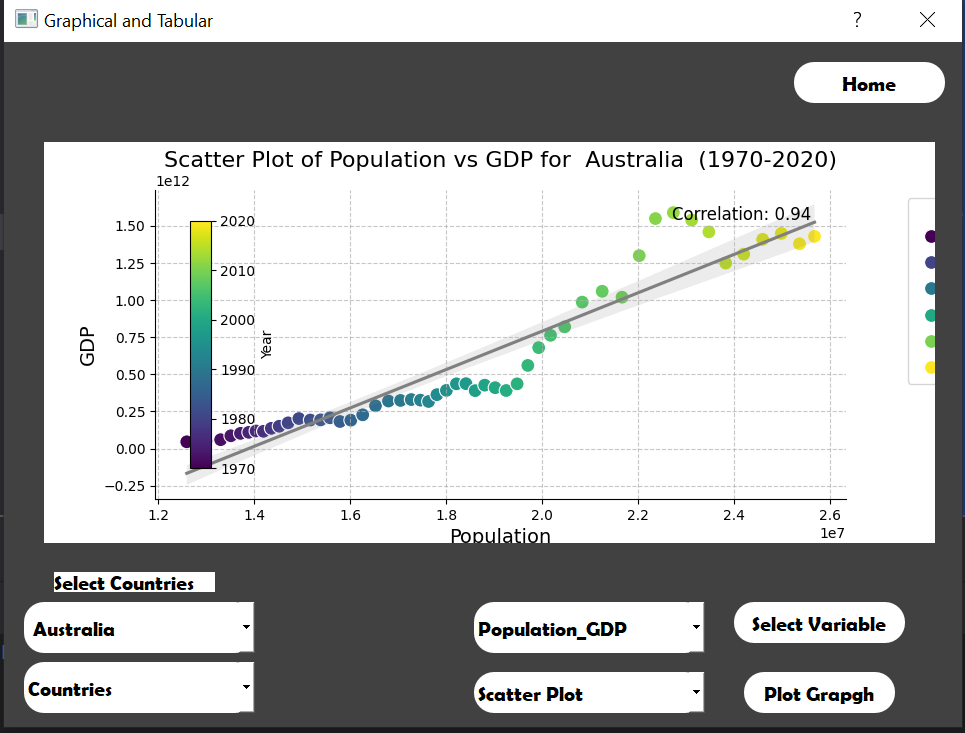
Main interface:



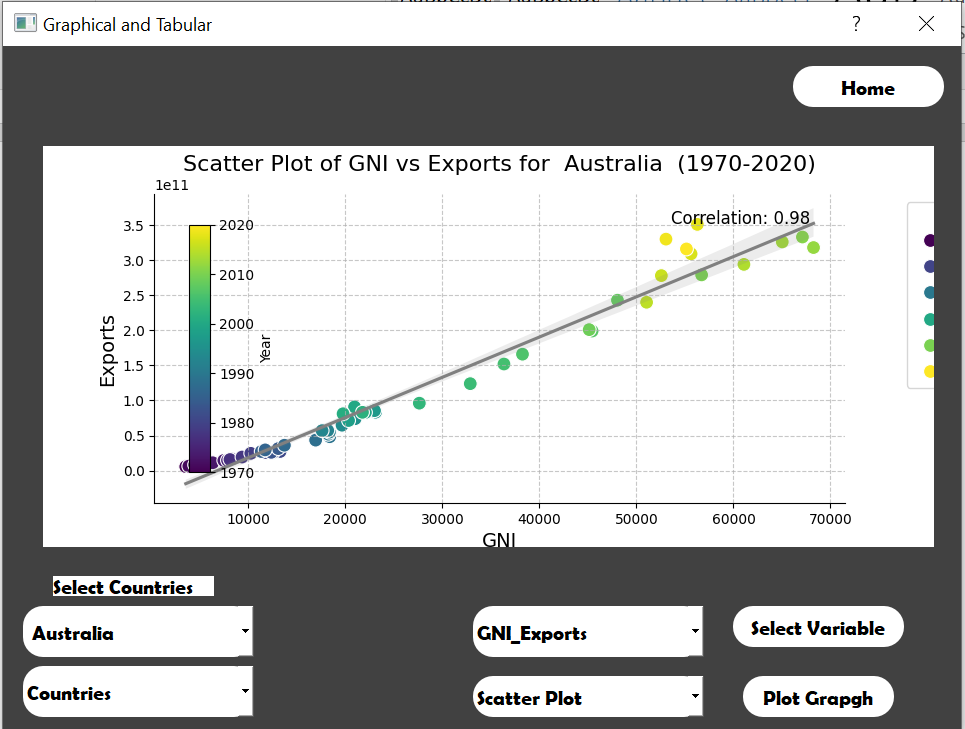
**Graphs:**



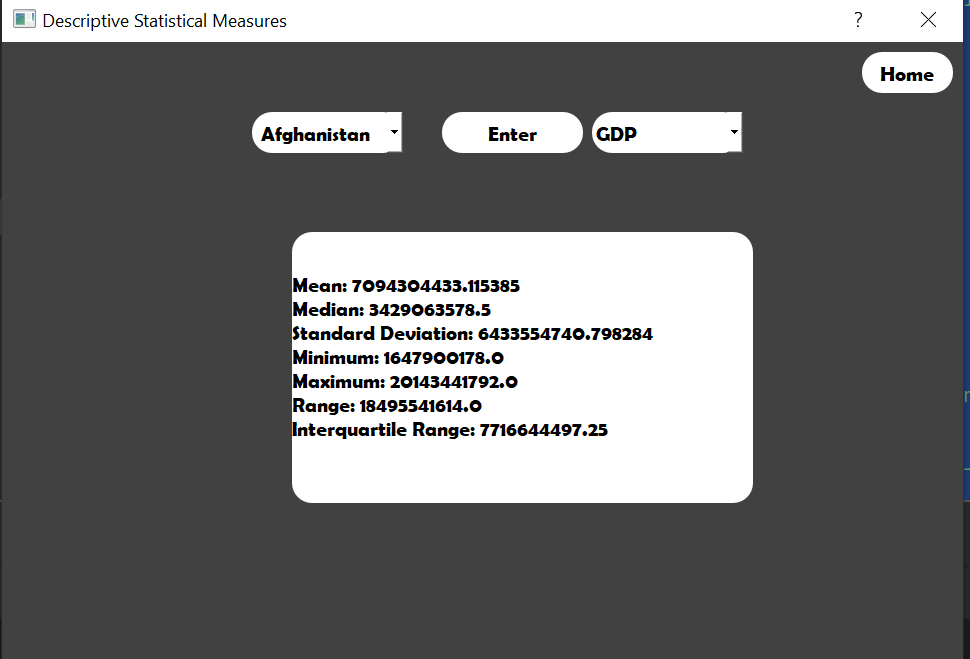


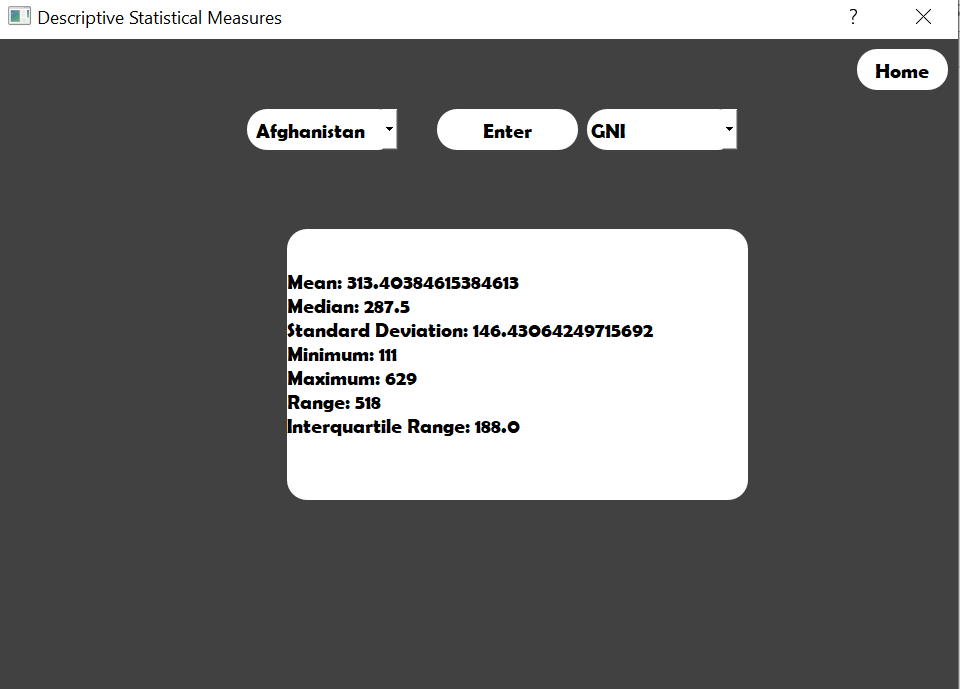


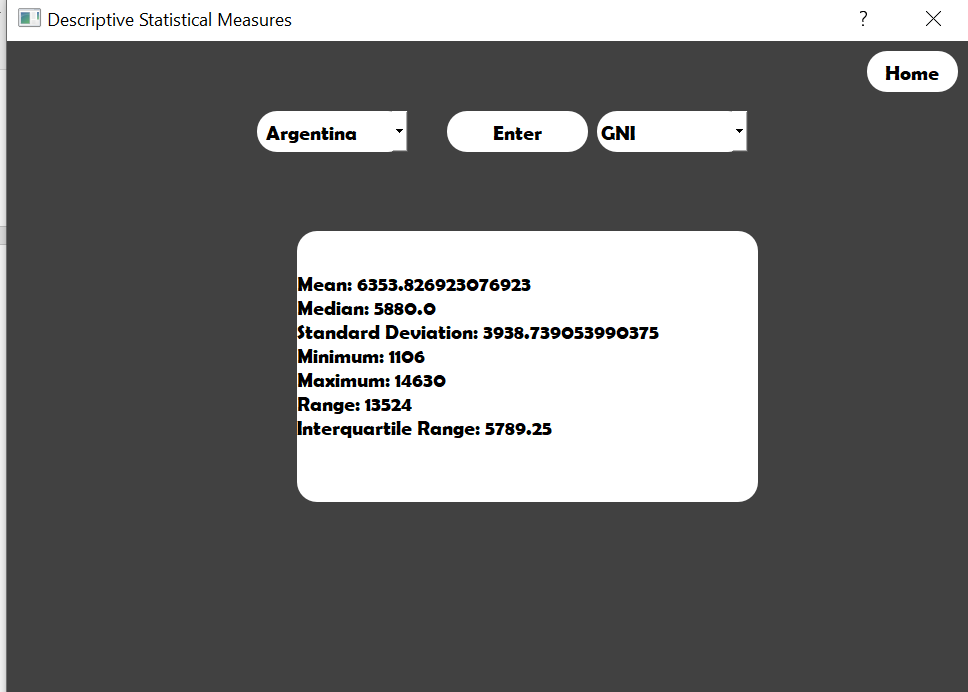


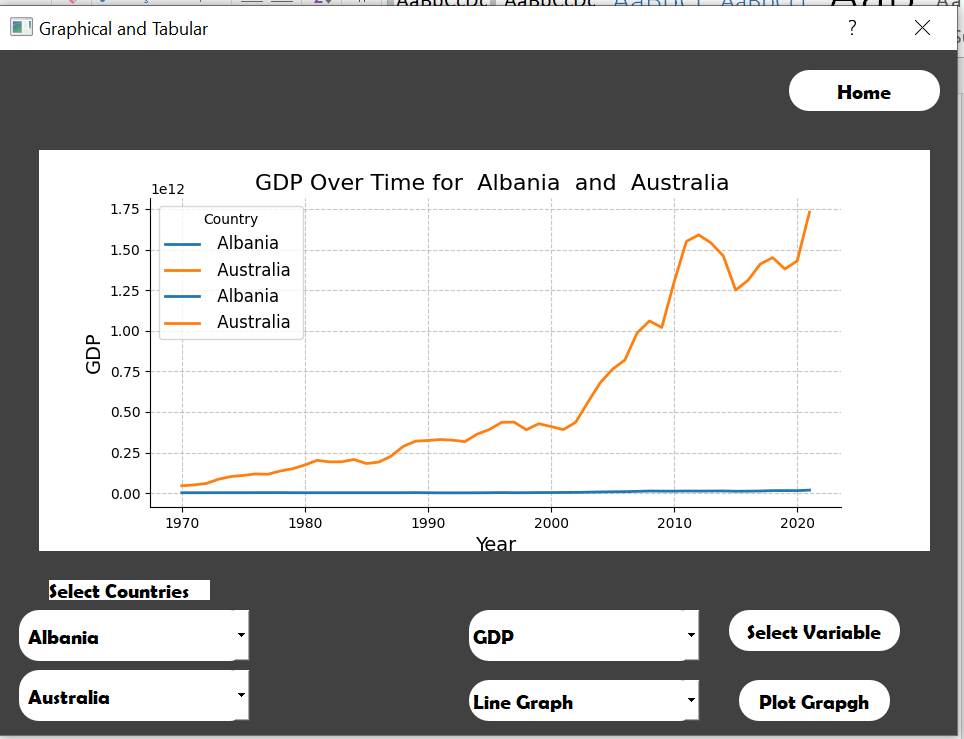


**Descriptive:**

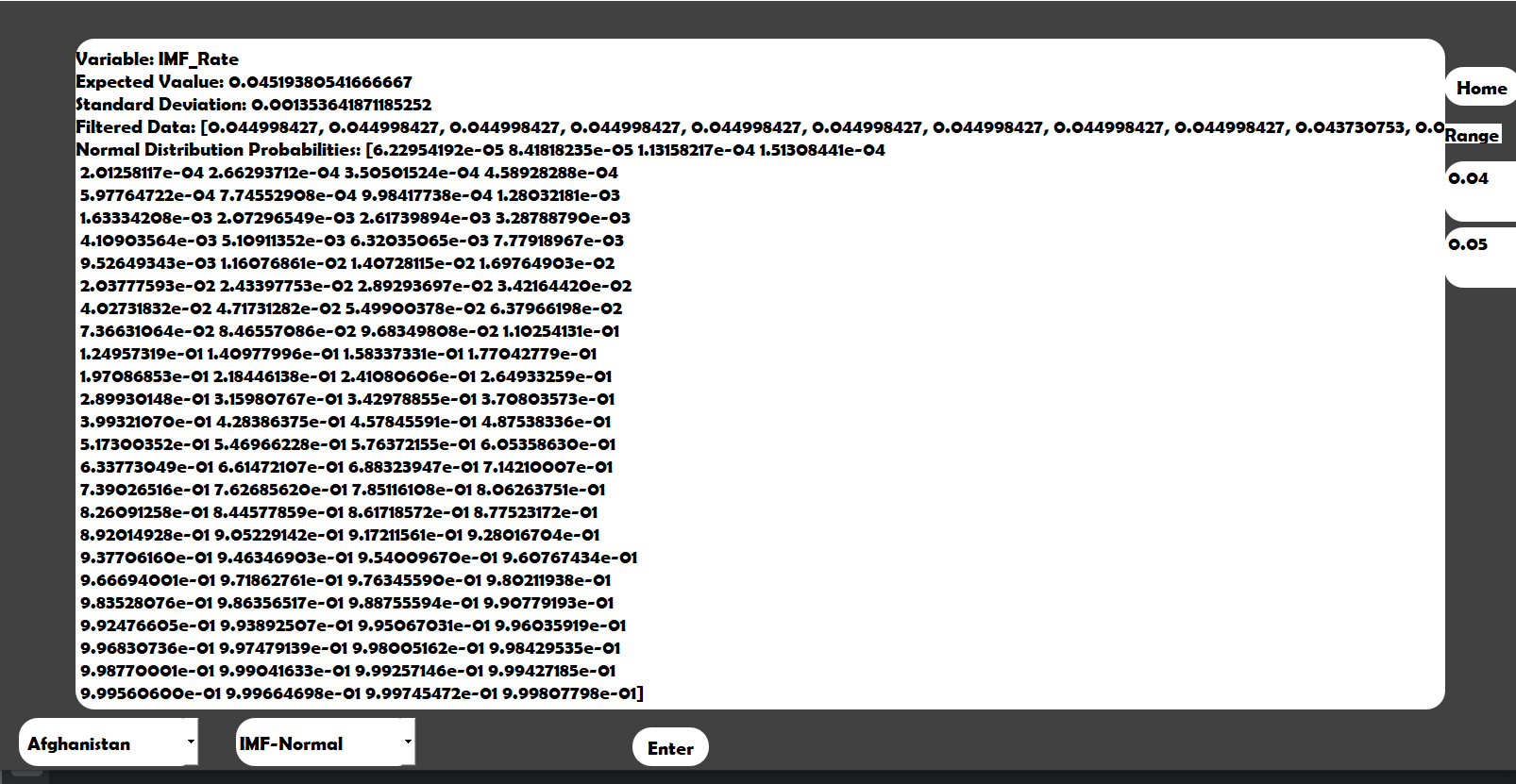




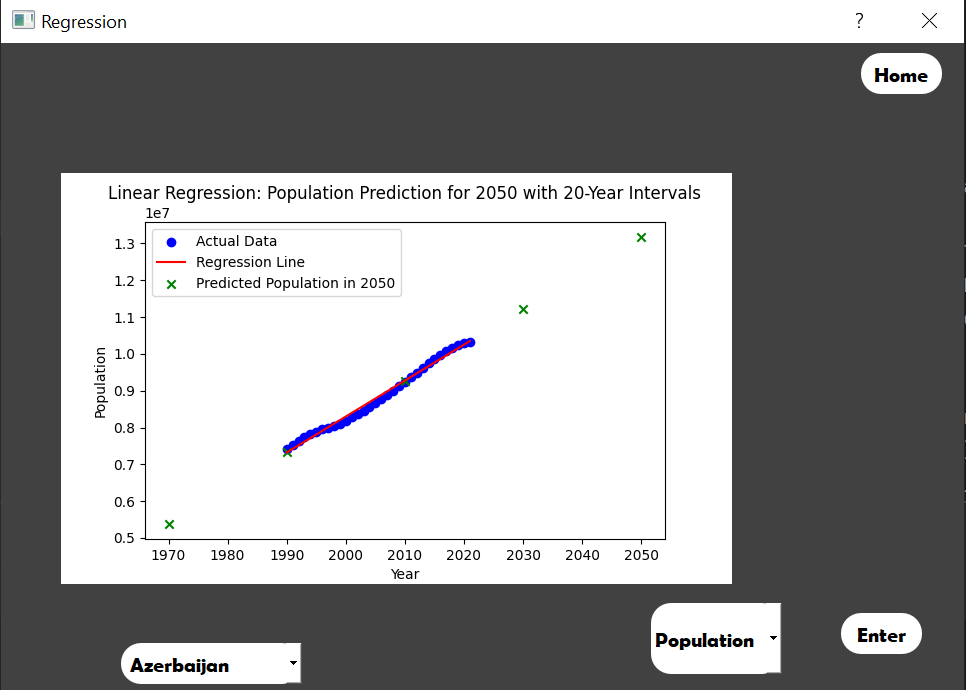


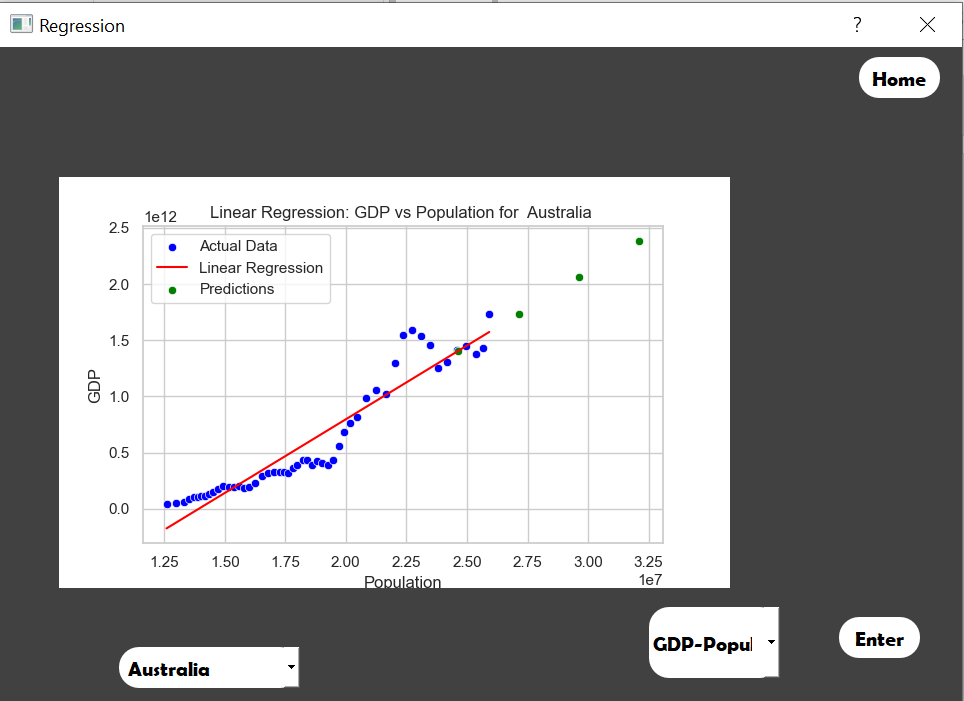


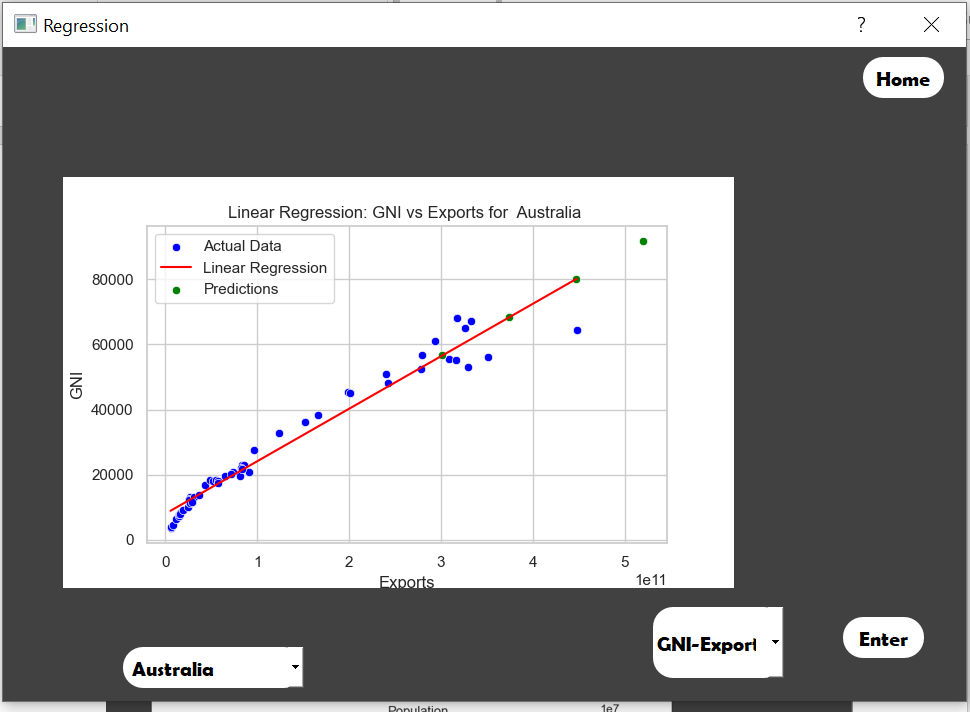
**Distribution:**



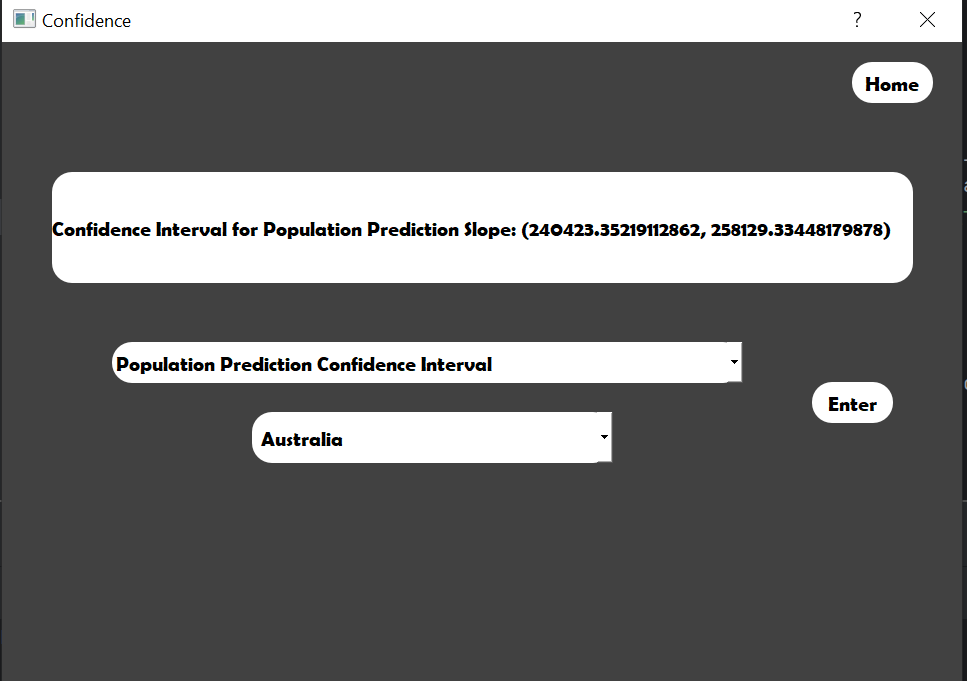
**Regression:**



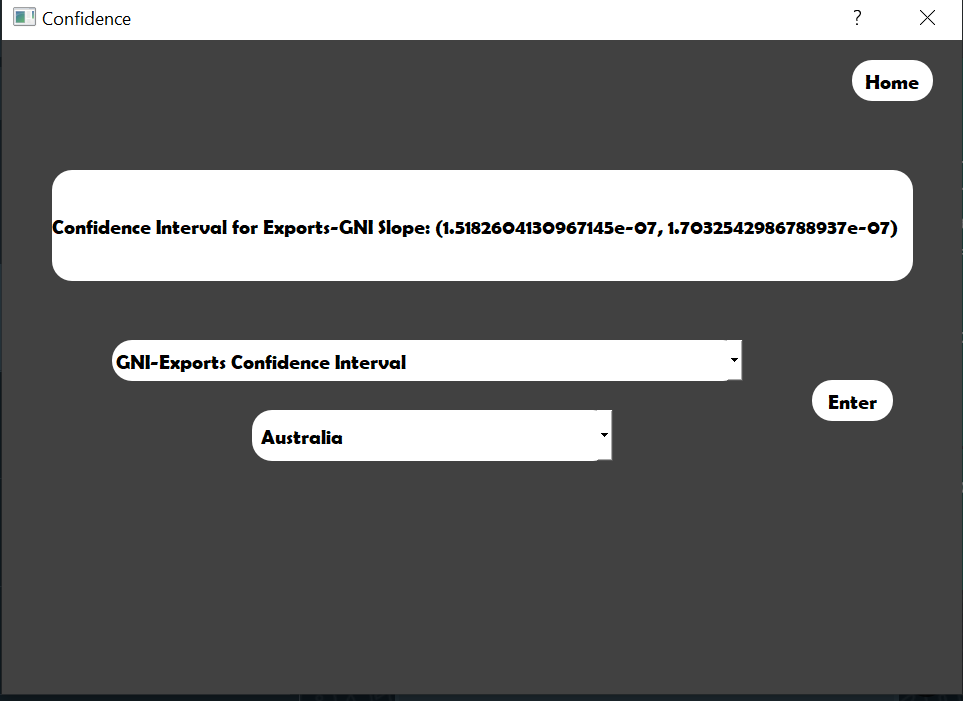




**Confidence:**







1. **Codes:**
2. from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog
3. from PyQt5.uic import loadUi
4. from PyQt5.QtWidgets import QComboBox
5. from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas
6. import seaborn as sns
7. import matplotlib.pyplot as plt
8. import pandas as pd
9. import matplotlib.cm as cm
10. import numpy as np
11. from statistics import mean, median, stdev
12. from sklearn.linear\_model import LinearRegression
13. from scipy.stats import norm, poisson, kstest
14. import scipy.stats as stats
15. from PyQt5.QtWidgets import QApplication, QLabel, QVBoxLayout, QScrollArea, QWidget
16. import numpy as np
17. df = pd.read\_csv('Global Economy Indicators.csv')
18. class MyGUI(QDialog):
19. def \_\_init\_\_(self):
20. super(MyGUI, self).\_\_init\_\_()
21. loadUi("untitled.ui", self)
22. self.pushButton\_7.clicked.connect(self.show\_graph)
23. self.pushButton\_3.clicked.connect(self.show\_descriptive)
24. self.pushButton\_4.clicked.connect(self.show\_probability)
25. self.pushButton\_5.clicked.connect(self.show\_regression)
26. self.pushButton\_6.clicked.connect(self.show\_confidence)
27. def show\_graph(self):
28. self.hide()
29. graph\_dialog = GraphDialog()
30. graph\_dialog.exec\_()
31. def show\_descriptive(self):
32. self.hide()
33. descriptive\_dialog = DescriptiveDialog()
34. descriptive\_dialog.exec\_()
35. def show\_probability(self):
36. self.hide()
37. probability\_dialog = ProbabilityDialog()
38. probability\_dialog.exec\_()
39. def show\_regression(self):
40. self.hide()
41. regression\_dialog = RegressionDialog()
42. regression\_dialog.exec\_()
43. def show\_confidence(self):
44. self.hide()
45. confidence\_dialog = ConfidenceDialog()
46. confidence\_dialog.exec\_()
47. class GraphDialog(QDialog):
48. def \_\_init\_\_(self):
49. super(GraphDialog, self).\_\_init\_\_()
50. loadUi("graph.ui", self)
51. self.country1 = ""
52. self.country2 = ""
53. self.f\_country()
54. self.selected\_countries\_data = [self.country1, self.country2]
55. self.enter.clicked.connect(self.my)
56. self.home.clicked.connect(self.Home)
57. self.e.clicked.connect(self.comboBoxg2\_changed)
58. self.comboBoxg.addItems(["AMA", "GDP", "Population\_GDP", "GNI\_Exports", 'Import\_Export'])
59. def Home(self):
60. self.hide()
61. mygui = MyGUI()
62. mygui.exec\_()
63. def my(self):
64. print("here again")
65. self.comboBoxg2.clear()
66. self.country1 = self.comboBoxg3.currentText()
67. self.country2 = self.comboBoxg4.currentText()
68. self.comboBoxg\_changed()
69. def comboBoxg\_changed(self):
70. self.selected\_item = self.comboBoxg.currentText()
71. if self.selected\_item == 'AMA':
72. self.comboBoxg2.addItems(["Options", "Histogram", "Line Graph"])
73. elif self.selected\_item == 'GDP':
74. self.comboBoxg2.addItems(["Options", "Line Graph", "Histogram"])
75. elif self.selected\_item == 'Population\_GDP' or self.selected\_item == 'GNI\_Exports':
76. self.comboBoxg2.addItems(["Options", "Scatter Plot"])
77. elif self.selected\_item == 'Import\_Export':
78. self.comboBoxg2.addItems(["Options", "Box Plot"])
79. def comboBoxg2\_changed(self):
80. var = self.comboBoxg.currentText()
81. graphName = self.comboBoxg2.currentText()
82. if graphName == "Histogram":
83. self.plot\_histogram()
84. elif graphName == "Line Graph":
85. self.plot\_Line\_graph()
86. elif graphName =="Scatter Plot":
87. self.scatter\_plot()
88. elif graphName =="Box Plot":
89. self.box\_plot()
90. def scatter\_plot(self):
91. if self.country1 != "Countries":
92. start\_year = 1970
93. end\_year = 2020
94. items = self.selected\_item.split('\_')
95. a, b = items
96. plt.figure(figsize=(12, 8))
97. data\_filtered = df[(df['Country'] == self.country1) & (df['Year'] >= start\_year) & (df['Year'] <= end\_year)]
98. plot1 = sns.scatterplot(x=a, y=b, hue='Year', palette='viridis', data=data\_filtered, s=100)
99. sns.regplot(x=a, y=b, data=data\_filtered, scatter=False, color='gray')
100. correlation\_coefficient = np.corrcoef(data\_filtered[a], data\_filtered[b])[0, 1]
101. correlation\_text = f'Correlation: {correlation\_coefficient:.2f}'
102. plt.text(0.95, 0.95, correlation\_text, transform=plt.gca().transAxes, ha='right', va='top', fontsize=12)
103. norm = plt.Normalize(data\_filtered['Year'].min(), data\_filtered['Year'].max())
104. sm = cm.ScalarMappable(cmap='viridis', norm=norm)
105. sm.set\_array([])
106. cax = plt.gca().inset\_axes([0.05, 0.1, 0.03, 0.8])
107. cbar = plt.colorbar(sm, cax=cax, label='Year',
108. orientation='vertical')
109. plt.title(f'Scatter Plot of {a} vs {b} for {self.country1} ({start\_year}-{end\_year})', fontsize=16)
110. plt.xlabel(a, fontsize=14)
111. plt.ylabel(b, fontsize=14)
112. handles1, labels1 = plot1.get\_legend\_handles\_labels()
113. plt.legend(handles=handles1, labels=labels1, title='Year', loc='upper right', bbox\_to\_anchor=(1.25, 1),
114. fontsize=12)
115. plt.grid(True, linestyle='--', alpha=0.7)
116. sns.despine()
117. canvas = FigureCanvas(plt.gcf())
118. for i in reversed(range(self.verticalLayoutg.count())):
119. self.verticalLayoutg.itemAt(i).widget().setParent(None)
120. self.verticalLayoutg.addWidget(canvas)
121. def plot\_Line\_graph(self):
122. if self.country1 != "Countries" and self.country2 != "Countries":
123. plt.figure(figsize=(12, 8))
124. plot1 = sns.lineplot(x='Year', y=self.selected\_item, data=df[df['Country'] == self.country1], label=self.country1,
125. linewidth=2, markers=True)
126. plot2 = sns.lineplot(x='Year', y=self.selected\_item, data=df[df['Country'] == self.country2], label=self.country2,
127. linewidth=2, markers=True)
128. plt.title(f'{self.selected\_item} Over Time for {self.country1} and {self.country2}', fontsize=16)
129. plt.xlabel('Year', fontsize=14)
130. plt.ylabel(f'{self.selected\_item}', fontsize=14)
131. handles1, labels1 = plot1.get\_legend\_handles\_labels()
132. handles2, labels2 = plot2.get\_legend\_handles\_labels()
133. plt.legend(handles=handles1 + handles2, labels=labels1 + labels2, title='Country', loc='upper left',
134. fontsize=12)
135. plt.grid(True, linestyle='--', alpha=0.7)
136. sns.despine()
137. canvas = FigureCanvas(plt.gcf())
138. for i in reversed(range(self.verticalLayoutg.count())):
139. self.verticalLayoutg.itemAt(i).widget().setParent(None)
140. self.verticalLayoutg.addWidget(canvas)
141. def box\_plot(self):
142. if self.country1 != "Countries":
143. plt.figure(figsize=(12, 8))
144. data\_country = df[df['Country'] == self.country1][['Year', 'Imports', 'Exports']]
145. data\_country\_melted = data\_country.melt(id\_vars=['Year'], var\_name='Transaction Type', value\_name='Value')
146. data\_country\_melted['Decade'] = (data\_country\_melted['Year'] // 10) \* 10
147. plot1 = sns.boxplot(x='Decade', y='Value', hue='Transaction Type', data=data\_country\_melted)
148. plt.title(f'Box Plot of Imports and Exports for {self.country1} (Grouped by Decades)', fontsize=16)
149. plt.xlabel('Decade', fontsize=14)
150. plt.ylabel('Value', fontsize=14)
151. plt.grid(True, linestyle='--', alpha=0.7)
152. sns.despine()
153. canvas = FigureCanvas(plt.gcf())
154. for i in reversed(range(self.verticalLayoutg.count())):
155. self.verticalLayoutg.itemAt(i).widget().setParent(None)
156. self.verticalLayoutg.addWidget(canvas)
157. def plot\_histogram(self):
158. if self.country1 != "Countries":
159. plt.figure(figsize=(12, 8))
160. data\_country = df[df['Country'] == self.country1][['Year', self.selected\_item]]
161. data\_country['Decade'] = (data\_country['Year'] // 10) \* 10
162. grouped\_data = data\_country.groupby('Decade')[self.selected\_item].mean()
163. plt.bar(grouped\_data.index, grouped\_data.values, width=8, align='edge', edgecolor='black')
164. plt.title(f'Histogram of {self.selected\_item} for {self.country1} (Year-wise with 10-Year Intervals)',
165. fontsize=16)
166. plt.xlabel('Decade', fontsize=14)
167. plt.ylabel(self.selected\_item, fontsize=14)
168. plt.grid(True, linestyle='-', alpha=0.7)
169. plt.gca().spines['top'].set\_visible(False)
170. plt.gca().spines['right'].set\_visible(False)
171. canvas = FigureCanvas(plt.gcf())
172. for i in reversed(range(self.verticalLayoutg.count())):
173. self.verticalLayoutg.itemAt(i).widget().setParent(None)
174. self.verticalLayoutg.addWidget(canvas)
175. class DescriptiveDialog(QDialog):
176. def \_\_init\_\_(self):
177. super(DescriptiveDialog, self).\_\_init\_\_()
178. loadUi("descriptive.ui", self)
179. self.country1 = ""
180. self.country2 = ""
181. self.h.clicked.connect(self.my)
182. self.home.clicked.connect(self.Home)
183. self.f\_country()
184. self.comboBoxd4.addItems(['GDP', 'AMA', 'GNI', 'Imports', 'Exports'])
185. def Home(self):
186. self.hide()
187. mygui = MyGUI()
188. mygui.exec\_()
189. def my(self):
190. self.country1 = self.comboBoxd3.currentText()
191. selected\_item = self.comboBoxd4.currentText()
192. if self.country1 != 'Countries' :
193. if selected\_item == 'GDP' or selected\_item =='ANI' or selected\_item =='GNI' or selected\_item =='Imports' or selected\_item =='Exports':
194. country\_data = df[df['Country'] == self.country1][selected\_item]
195. mean\_value = np.mean(country\_data)
196. median\_value = np.median(country\_data)
197. std\_dev\_value = np.std(country\_data)
198. min\_value = np.min(country\_data)
199. max\_value = np.max(country\_data)
200. range\_value = np.ptp(country\_data)
201. iqr = np.percentile(country\_data, 75) - np.percentile(country\_data, 25)
202. skewness = country\_data.skew()
203. kurtosis = country\_data.kurtosis()
204. label\_text = (
205. f'Mean: {mean\_value}\n'
206. f'Median: {median\_value}\n'
207. f'Standard Deviation: {std\_dev\_value}\n'
208. f'Minimum: {min\_value}\n'
209. f'Maximum: {max\_value}\n'
210. f'Range: {range\_value}\n'
211. f'Interquartile Range: {iqr}\n'
212. )
213. self.label.setText(label\_text)
214. self.label.show()
215. class ProbabilityDialog(QDialog):
216. def \_\_init\_\_(self):
217. super(ProbabilityDialog, self).\_\_init\_\_()
218. loadUi("Prob.ui", self)
219. self.f\_country()
220. self.country1 = ""
221. self.comboBoxp2.addItems(['Options', 'IMF-Normal', 'Exports-Uniform'])
222. self.e.clicked.connect(self.my)
223. def Home(self):
224. self.hide()
225. mygui = MyGUI()
226. mygui.exec\_()
227. def my(self):
228. self.country1 = self.comboBoxp.currentText()
229. selected\_item = self.comboBoxp2.currentText()
230. low = float(self.textEdit.toPlainText())
231. high = float(self.textEdit\_2.toPlainText())
232. print(low)
233. if self.country1 != 'Countries':
234. country\_data = df[df['Country'] == self.country1]
235. if selected\_item == 'Exports-Uniform':
236. self.analyze\_uniform\_distribution('Exports', country\_data, 0, 5000000000)
237. elif selected\_item == 'Changes\_in\_inventories':
238. self.analyze\_poisson\_distribution('Changes\_in\_inventories',country\_data)
239. elif selected\_item == 'IMF-Normal':
240. self.analyze\_normal\_distribution('IMF\_Rate', country\_data, low, high)
241. def analyze\_normal\_distribution(self, variable, country\_data, range\_low=None, range\_high=None):
242. data = country\_data[variable]
243. if range\_low is None:
244. range\_low = min(data)
245. if range\_high is None:
246. range\_high = max(data)
247. filtered\_data = [value for value in data if range\_low <= value <= range\_high]
248. mean, std = np.mean(filtered\_data), np.std(filtered\_data)
249. normal\_values = np.linspace(range\_low, range\_high, 100)
250. normal\_cumulative\_probabilities = stats.norm.cdf(normal\_values, mean, std)
251. self.label.setText(f'Variable: {variable}\n'
252. f'Expected Vaalue: {mean}\n'
253. f'Standard Deviation: {std}\n'
254. f'Filtered Data: {filtered\_data}\n'
255. f'Normal Distribution Probabilities: {normal\_cumulative\_probabilities}')
256. def analyze\_uniform\_distribution(self, variable, country\_data, range\_low=None, range\_high=None):
257. data = country\_data[variable]
258. if range\_low is None:
259. range\_low = min(data)
260. if range\_high is None:
261. range\_high = max(data)
262. filtered\_data = [value for value in data if range\_low <= value <= range\_high]
263. mean = (range\_low + range\_high) / 2
264. range\_diff = range\_high - range\_low
265. result\_text = (f'Variable: {variable}\n'
266. f'Mean: {mean}\n'
267. f'Range: {range\_low} to {range\_high}\n'
268. f'Filtered Data: {filtered\_data}\n'
269. f'Uniform Distribution Probability Density: 1 / {range\_diff}')
270. self.label.setText(result\_text)
271. def analyze\_poisson\_distribution(self, variable, country\_data, range\_low=None, range\_high=None):
272. data = country\_data[variable]
273. if range\_low is None:
274. range\_low = min(data)
275. if range\_high is None:
276. range\_high = max(data)
277. poisson\_lambda = np.mean(data)
278. poisson\_values = np.arange(range\_low, range\_high + 1)
279. poisson\_probabilities = stats.poisson.pmf(poisson\_values, poisson\_lambda)
280. plt.hist(data, density=True, alpha=0.6, color='g', label='Histogram')
281. plt.plot(poisson\_values, poisson\_probabilities, 'r--', label='Poisson Distribution')
282. plt.title(f'Poisson Distribution Analysis for {variable}')
283. plt.xlabel(variable)
284. plt.ylabel('Probability Density')
285. plt.legend()
286. plt.show()
287. def f\_country(self):
288. self.comboBoxp.addItems(['Countries', ' Afghanistan ', ' Albania ', ' Algeria ', ' Andorra ', ' Angola ',
289. ' Antigua and Barbuda ', ' Azerbaijan ', ' Argentina ',
290. ' Australia ', ' Austria ', ' Bahamas ', ' Bahrain ',
291. ' Bangladesh ', ' Armenia ', ' Barbados ', ' Belgium ',
292. ' Bermuda ', ' Bhutan ', ' Bolivia (Plurinational State of) ',
293. ' Bosnia and Herzegovina ', ' Botswana ', ' Brazil ', ' Belize ',
294. ' Solomon Islands ', ' British Virgin Islands ',
295. ' Brunei Darussalam ', ' Bulgaria ', ' Myanmar ', ' Burundi ',
296. ' Belarus ', ' Cambodia ', ' Cameroon ', ' Canada ',
297. ' Cabo Verde ', ' Cayman Islands ', ' Central African Republic ',
298. ' Sri Lanka ', ' Chad ', ' Chile ', ' China ', ' Colombia ',
299. ' Comoros ', ' Congo ', ' D.R. of the Congo ', ' Cook Islands ',
300. ' Costa Rica ', ' Croatia ', ' Cuba ', ' Cyprus ',
301. ' Czechoslovakia (Former) ', ' Czechia ', ' Benin ', ' Denmark ',
302. ' Dominica ', ' Dominican Republic ', ' Ecuador ', ' El Salvador ',
303. ' Equatorial Guinea ', ' Ethiopia (Former) ', ' Ethiopia ',
304. ' Eritrea ', ' Estonia ', ' Fiji ', ' Finland ', ' France ',
305. ' French Polynesia ', ' Djibouti ', ' Gabon ', ' Georgia ',
306. ' Gambia ', ' State of Palestine ', ' Germany ', ' Ghana ',
307. ' Kiribati ', ' Greece ', ' Greenland ', ' Grenada ',
308. ' Guatemala ', ' Guinea ', ' Guyana ', ' Haiti ', ' Honduras ',
309. ' China, Hong Kong SAR ', ' Hungary ', ' Iceland ', ' India ',
310. ' Indonesia ', ' Iran (Islamic Republic of) ', ' Iraq ',
311. ' Ireland ', ' Israel ', ' Italy ', " Côte d'Ivoire ", ' Jamaica ',
312. ' Japan ', ' Kazakhstan ', ' Jordan ', ' Kenya ',
313. ' D.P.R. of Korea ', ' Republic of Korea ', ' Kosovo ', ' Kuwait ',
314. ' Kyrgyzstan ', " Lao People's DR ", ' Lebanon ', ' Lesotho ',
315. ' Latvia ', ' Liberia ', ' Libya ', ' Liechtenstein ',
316. ' Lithuania ', ' Luxembourg ', ' China, Macao SAR ',
317. ' Madagascar ', ' Malawi ', ' Malaysia ', ' Maldives ', ' Mali ',
318. ' Malta ', ' Mauritania ', ' Mauritius ', ' Mexico ', ' Monaco ',
319. ' Mongolia ', ' Republic of Moldova ', ' Montenegro ',
320. ' Montserrat ', ' Morocco ', ' Mozambique ', ' Oman ', ' Namibia ',
321. ' Nauru ', ' Nepal ', ' Netherlands ',
322. ' Former Netherlands Antilles ', ' Aruba ', ' New Caledonia ',
323. ' Vanuatu ', ' New Zealand ', ' Nicaragua ', ' Niger ',
324. ' Nigeria ', ' Norway ', ' Micronesia (FS of) ',
325. ' Marshall Islands ', ' Palau ', ' Pakistan ', ' Panama ',
326. ' Papua New Guinea ', ' Paraguay ', ' Peru ', ' Philippines ',
327. ' Poland ', ' Portugal ', ' Guinea-Bissau ', ' Timor-Leste ',
328. ' Puerto Rico ', ' Qatar ', ' Romania ', ' Russian Federation ',
329. ' Rwanda ', ' Saint Kitts and Nevis ', ' Anguilla ',
330. ' Saint Lucia ', ' St. Vincent and the Grenadines ',
331. ' San Marino ', ' Sao Tome and Principe ', ' Saudi Arabia ',
332. ' Senegal ', ' Serbia ', ' Seychelles ', ' Sierra Leone ',
333. ' Singapore ', ' Slovakia ', ' Viet Nam ', ' Slovenia ',
334. ' Somalia ', ' South Africa ', ' Zimbabwe ',
335. ' Yemen Democratic (Former) ', ' Spain ', ' Sudan (Former) ',
336. ' Suriname ', ' Eswatini ', ' Sweden ', ' Switzerland ',
337. ' Syrian Arab Republic ', ' Tajikistan ', ' Thailand ', ' Togo ',
338. ' Tonga ', ' Trinidad and Tobago ', ' United Arab Emirates ',
339. ' Tunisia ', ' Türkiye ', ' Turkmenistan ',
340. ' Turks and Caicos Islands ', ' Tuvalu ', ' Uganda ', ' Ukraine ',
341. ' North Macedonia ', ' USSR (Former) ', ' Egypt ',
342. ' United Kingdom ', ' U.R. of Tanzania: Mainland ', ' Zanzibar ',
343. ' United States ', ' Burkina Faso ', ' Uruguay ', ' Uzbekistan ',
344. ' Venezuela (Bolivarian Republic of) ', ' Samoa ',
345. ' Yemen Arab Republic (Former) ', ' Yemen ',
346. ' Yugoslavia (Former) ', ' Zambia ', ' Curaçao ',
347. ' Sint Maarten (Dutch part) ', ' South Sudan ', ' Sudan '])
348. class RegressionDialog(QDialog):
349. def \_\_init\_\_(self):
350. super(RegressionDialog, self).\_\_init\_\_()
351. loadUi("regression.ui", self)
352. self.f\_country()
353. self.country1 = ""
354. self.comboBoxr.addItems(['Options', 'Population Prediction', 'GDP-Population', 'GNI-Exports'])
355. self.b.clicked.connect(self.my)
356. def my(self):
357. selected\_item = self.comboBoxr.currentText()
358. self.country1 = self.comboBoxr2.currentText()
359. if self.country1 !='Countries':
360. if selected\_item == 'Population Prediction':
361. country\_data = df[df['Country'] == self.country1]
362. years = country\_data['Year'].values.reshape(-1, 1)
363. population = country\_data['Population'].values.reshape(-1, 1)
364. model = LinearRegression()
365. model.fit(years, population)
366. future\_years = np.arange(1970, 2051, 20).reshape(-1, 1)
367. predicted\_population\_2050 = model.predict(future\_years)
368. plt.figure(figsize=(12, 8))
369. plt.scatter(years, population, label='Actual Data', color='blue')
370. plt.plot(years, model.predict(years), label='Regression Line', color='red')
371. plt.scatter(future\_years, predicted\_population\_2050, color='green', marker='x',
372. label='Predicted Population in 2050')
373. plt.xlabel('Year')
374. plt.ylabel('Population')
375. plt.title('Linear Regression: Population Prediction for 2050 with 20-Year Intervals')
376. plt.legend()
377. canvas = FigureCanvas(plt.gcf())
378. for i in reversed(range(self.verticalLayoutr.count())):
379. self.verticalLayoutr.itemAt(i).widget().setParent(None)
380. self.verticalLayoutr.addWidget(canvas)
381. canvas.draw()
382. elif selected\_item =='GDP-Population':
383. country\_data = df[df['Country'] == self.country1]
384. if country\_data.empty:
385. raise ValueError(f"No data found for {self.country1}")
386. gdp = country\_data['GDP'].values.reshape(-1, 1)
387. population = country\_data['Population'].values.reshape(-1, 1)
388. country\_data = df[df['Country'] ==self.country1]
389. years = country\_data['Year'].values.reshape(-1, 1)
390. population = country\_data['Population'].values.reshape(-1, 1)
391. model1 = LinearRegression()
392. model1.fit(years, population)
393. future\_years = np.arange(2020, 2051, 10).reshape(-1, 1)
394. predicted\_population = model1.predict(future\_years)
395. population\_2020 = predicted\_population[0][0]
396. population\_2030 = predicted\_population[1][0]
397. population\_2040 = predicted\_population[2][0]
398. population\_2050 = predicted\_population[3][0]
399. model = LinearRegression()
400. model.fit(population, gdp)
401. new\_population\_values = np.array([population\_2020, population\_2030, population\_2040, population\_2050])
402. new\_population\_values = new\_population\_values.reshape(-1, 1)
403. predicted\_gdp = model.predict(new\_population\_values)
404. sns.set(style="whitegrid")
405. plt.figure(figsize=(10, 6))
406. sns.scatterplot(x=population.flatten(), y=gdp.flatten(), color='blue', label='Actual Data')
407. sns.lineplot(x=population.flatten(), y=model.predict(population).flatten(), color='red',
408. label='Linear Regression')
409. sns.scatterplot(x=new\_population\_values.flatten(), y=predicted\_gdp.flatten(), color='green',
410. label='Predictions')
411. plt.xlabel('Population')
412. plt.ylabel('GDP')
413. plt.title(f'Linear Regression: GDP vs Population for {self.country1}')
414. plt.legend()
415. canvas = FigureCanvas(plt.gcf())
416. for i in reversed(range(self.verticalLayoutr.count())):
417. self.verticalLayoutr.itemAt(i).widget().setParent(None)
418. self.verticalLayoutr.addWidget(canvas)
419. canvas.draw()
420. elif selected\_item == 'GNI-Exports':
421. country\_data = df[df['Country'] == self.country1]
422. if country\_data.empty:
423. raise ValueError(f"No data found for {self.country1}")
424. gdp = country\_data['GNI'].values.reshape(-1, 1)
425. population = country\_data['Exports'].values.reshape(-1, 1)
426. country\_data = df[df['Country'] == self.country1]
427. years = country\_data['Year'].values.reshape(-1, 1)
428. population = country\_data['Exports'].values.reshape(-1, 1)
429. model1 = LinearRegression()
430. model1.fit(years, population)
431. future\_years = np.arange(2020, 2051, 10).reshape(-1, 1)
432. predicted\_population = model1.predict(future\_years)
433. population\_2020 = predicted\_population[0][0]
434. population\_2030 = predicted\_population[1][0]
435. population\_2040 = predicted\_population[2][0]
436. population\_2050 = predicted\_population[3][0]
437. model = LinearRegression()
438. model.fit(population, gdp)
439. new\_population\_values = np.array([population\_2020, population\_2030, population\_2040, population\_2050])
440. new\_population\_values = new\_population\_values.reshape(-1, 1)
441. predicted\_gdp = model.predict(new\_population\_values)
442. sns.set(style="whitegrid")
443. plt.figure(figsize=(10, 6))
444. sns.scatterplot(x=population.flatten(), y=gdp.flatten(), color='blue', label='Actual Data')
445. sns.lineplot(x=population.flatten(), y=model.predict(population).flatten(), color='red',
446. label='Linear Regression')
447. sns.scatterplot(x=new\_population\_values.flatten(), y=predicted\_gdp.flatten(), color='green',
448. label='Predictions')
449. plt.xlabel('Exports')
450. plt.ylabel('GNI')
451. plt.title(f'Linear Regression: GNI vs Exports for {self.country1}')
452. plt.legend()
453. canvas = FigureCanvas(plt.gcf())
454. for i in reversed(range(self.verticalLayoutr.count())):
455. self.verticalLayoutr.itemAt(i).widget().setParent(None)
456. self.verticalLayoutr.addWidget(canvas)
457. canvas.draw()
458. class ConfidenceDialog(QDialog):
459. def \_\_init\_\_(self):
460. super(ConfidenceDialog, self).\_\_init\_\_()
461. loadUi("Confidence.ui", self)
462. self.f\_country()
463. self.comboBoxc.addItems(['Options', 'Population Prediction Confidence Interval',
464. 'GDP-Population Confidence Interval', 'GNI-Exports Confidence Interval'])
465. self.b.clicked.connect(self.calculate\_confidence\_interval)
466. def calculate\_confidence\_interval(self):
467. print("A")
468. selected\_item = self.comboBoxc.currentText()
469. country = self.c.currentText()
470. if selected\_item == 'Population Prediction Confidence Interval':
471. country\_data = df[df['Country'] == country]
472. years = country\_data['Year'].values.reshape(-1, 1)
473. population = country\_data['Population'].values.reshape(-1, 1)
474. model = LinearRegression()
475. model.fit(years, population)
476. slope, intercept = model.coef\_[0][0], model.intercept\_[0]
477. y\_pred = model.predict(years)
478. residuals = population.flatten() - y\_pred.flatten()
479. mse = np.sum(residuals \*\* 2) / len(residuals)
480. std\_err\_slope = np.sqrt(mse / np.sum((years - np.mean(years)) \*\* 2))
481. t\_value = 1.96
482. margin\_error = t\_value \* std\_err\_slope
483. lower\_bound = slope - margin\_error
484. upper\_bound = slope + margin\_error
485. self.label.setText(f"Confidence Interval for Population Prediction Slope: ({lower\_bound}, {upper\_bound})")
486. self.label.show()
487. elif selected\_item == 'GDP-Population Confidence Interval':
488. country\_data = df[df['Country'] == country]
489. years = country\_data['Year'].values.reshape(-1, 1)
490. population = country\_data['Population'].values.reshape(-1, 1)
491. gdp = country\_data['GDP'].values.reshape(-1, 1)
492. model\_population = LinearRegression()
493. model\_population.fit(years, population)
494. model\_gdp = LinearRegression()
495. model\_gdp.fit(population, gdp)
496. def calculate\_interval(model, x, y):
497. y\_pred = model.predict(x)
498. residuals = y.flatten() - y\_pred.flatten()
499. mse = np.sum(residuals \*\* 2) / len(residuals)
500. std\_err\_slope = np.sqrt(mse / np.sum((x - np.mean(x)) \*\* 2))
501. t\_value = 1.96
502. margin\_error = t\_value \* std\_err\_slope
503. slope, intercept = model.coef\_[0][0], model.intercept\_[0]
504. lower\_bound = slope - margin\_error
505. upper\_bound = slope + margin\_error
506. return lower\_bound, upper\_bound
507. lower\_bound\_gdp, upper\_bound\_gdp = calculate\_interval(model\_gdp, population, gdp)
508. self.label.setText(f"Confidence Interval for GDP-Population Slope: ({lower\_bound\_gdp}, {upper\_bound\_gdp})")
509. self.label.show()
510. elif selected\_item == 'GNI-Exports Confidence Interval':
511. country\_data = df[df['Country'] == country]
512. years = country\_data['Year'].values.reshape(-1, 1)
513. gni = country\_data['GNI'].values.reshape(-1, 1)
514. exports = country\_data['Exports'].values.reshape(-1, 1)
515. model\_gni = LinearRegression()
516. model\_gni.fit(years, gni)
517. model\_exports = LinearRegression()
518. model\_exports.fit(exports, gni)
519. def calculate\_interval(model, x, y):
520. y\_pred = model.predict(x)
521. residuals = y.flatten() - y\_pred.flatten()
522. mse = np.sum(residuals \*\* 2) / len(residuals)
523. std\_err\_slope = np.sqrt(mse / np.sum((x - np.mean(x)) \*\* 2))
524. t\_value = 1.96
525. margin\_error = t\_value \* std\_err\_slope
526. slope, intercept = model.coef\_[0][0], model.intercept\_[0]
527. lower\_bound = slope - margin\_error
528. upper\_bound = slope + margin\_error
529. return lower\_bound, upper\_bound
530. lower\_bound\_exports, upper\_bound\_exports = calculate\_interval(model\_exports, exports, gni)
531. self.label.setText(f"Confidence Interval for Exports-GNI Slope: ({lower\_bound\_exports}, {upper\_bound\_exports})")
532. self.label.show()
533. def main():
534. app = QApplication([])
535. window = MyGUI()
536. window.show()
537. app.exec\_()
538. """
539. def fit\_distribution(data, column\_name):
540. if "Agriculture" in column\_name:
541. distribution = poisson
542. else:
543. distribution = norm
544. # Fit the distribution to the data
545. parameters = distribution.fit(data)
546. # Print the distribution name
547. print(f"Distribution applied to {column\_name}: {distribution.name}")
548. """
549. if \_\_name\_\_ == '\_\_main\_\_':
550. main()
551. """
552. country\_name = ' Pakistan '
553. fit\_distribution(df['Population'], 'Population')
554. fit\_distribution(df['Gross\_National\_Income(GNI)\_in\_USD'], 'GNI')
555. fit\_distribution(df['GDP'], 'GDP')
556. fit\_distribution(df['Agriculture\_hunting\_forestry\_fishing\_(ISIC\_A-B)'], 'Agriculture')
557. fit\_distribution(df['Imports'], 'Imports')
558. fit\_distribution(df['Exports'], 'Exports')
559. fit\_distribution(df['Manufacturing\_(ISIC\_D)'], 'Manufacturing')
560. """
561. **Conclusion:**

In wrapping up this project, we've thoroughly looked at economic data from **1970 to 2021** for different countries. We explored important stuff like **GDP**, **Exports and imports**, **population**, **income**, and overall economic output. Using all formats, we made educated guesses about what might happen to these countries in the future. Comparing different countries helped us see what makes each one unique economically. We plan to keep improving our methods as we learn more and get better data, ensuring our predictions stay reliable over time.