

Web Applications for Cost Analysis Use: R Shiny vs Python Plotly Dash

Eric J. Hagee
John W. Maddrey



Outline

- Introduction to Web Apps
- Project Description and Term Definition
- R Shiny vs Python Dash
 - Community Usage and Resources
 - File Structure
 - Layout, Styling, and Graphics
 - Reactivity
 - Enterprise Deployment
- Conclusions



What are Web Apps and Why do we Care?

- The availability of data and increase in computer power has created interest in collation of data, often in “real-time”
- Web browsers use modern programming languages to render information in html format like web pages
 - Transforms code into html, rendered as a web page
 - Can present data and information with custom visualizations and annotation
 - Use of programming languages can deftly handle data and produce sophisticated graphics
 - **Web Applications can be viewed by multiple users in a web browser without any other applications**



General Architecture

End Users

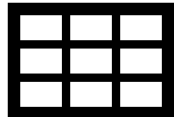
UI

Server

The UI displays data input and input parameters widgets



End Users upload data to UI and select Input Parameters



Server loads packages



UI sends data and input parameters to Server



Server analyzes data, performs any transformations, and generates tables and graphs



UI displays outputs



Users view and/or download outputs



Project Description

- The programming languages R and Python are the two main open source data environments
- Each has packages that enable creation of web apps
 - R – Shiny
 - Python – Plotly Dash
- Created an inflation application in each package to illustrate comparisons between languages and packages
 - The intent was to demonstrate basic capabilities, so app is simple by design
 - Public inflation data is readily available and relevant to wider discussions and specifically to cost analysis
 - The application has several pages that discuss inflation definitions and inflation measures and graphs of historical data, as well as an inflation calculator and relevant reference links
 - The code utilizes the same basic format and calculations, but differs in syntax and any package-specific organization



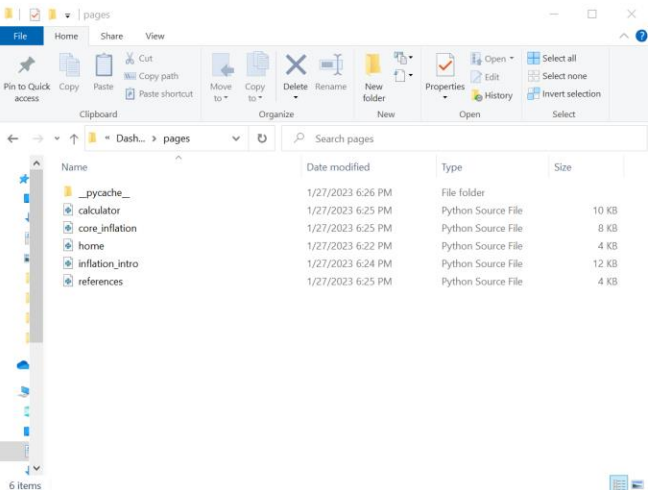
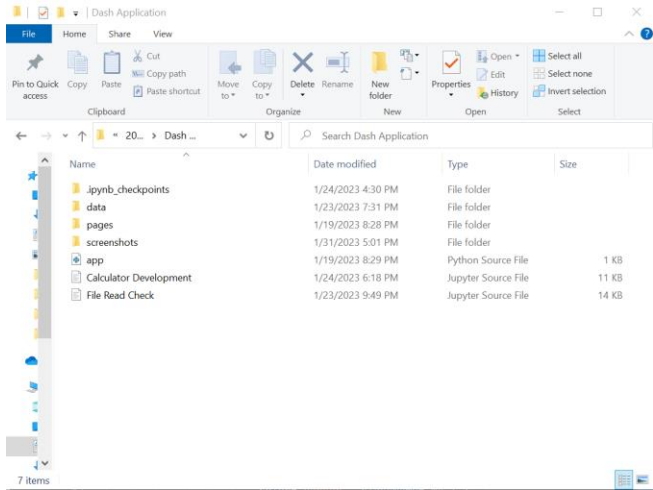
Comparisons – Community and General Considerations

- Community:
 - R Shiny was founded the same year as Plotly, out of an existing effort
 - R Shiny has more historical usage and is thus more established
 - More community resources on for example, Stack Exchange
 - Questions 6 to 1 on Stack Exchange
 - Numbers are more even on recent questions
- Documentation:
 - Good Documentation on Both
 - R Shiny has better introduction and lots of articles
 - Python Dash has more intermediate specific examples that and more organized function documentation
- Language Usage:
 - Python is more widely used than R (5 to 1 questions on Stack Exchange)
 - Python is more of a programming language than R, but R is made for statistical modeling
- The Twist!: Both Shiny and Dash are developing a corresponding Python and/or R version

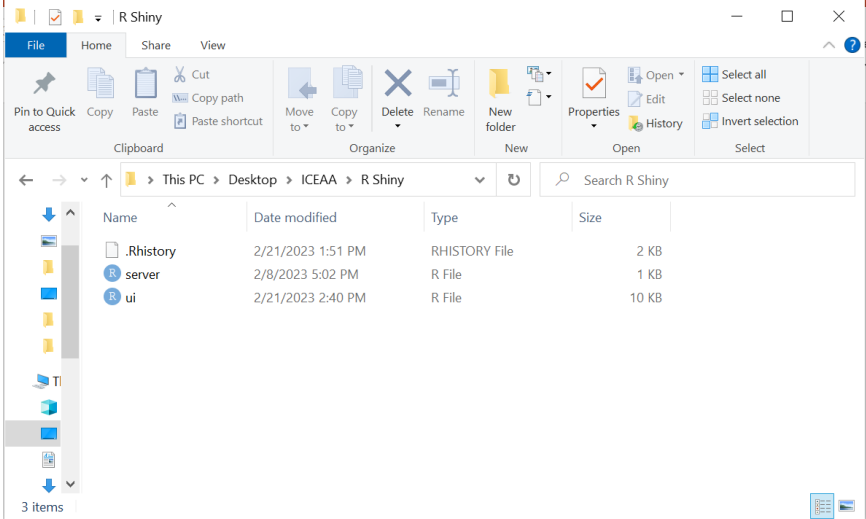


Comparisons – File Structure

Python Dash File Structure



R Shiny File Structure



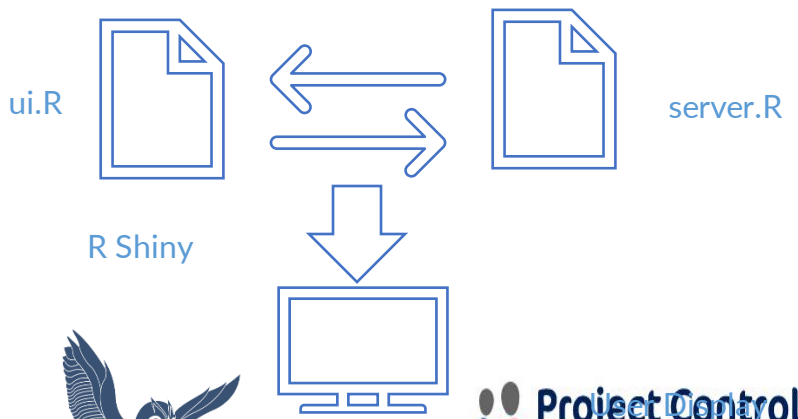
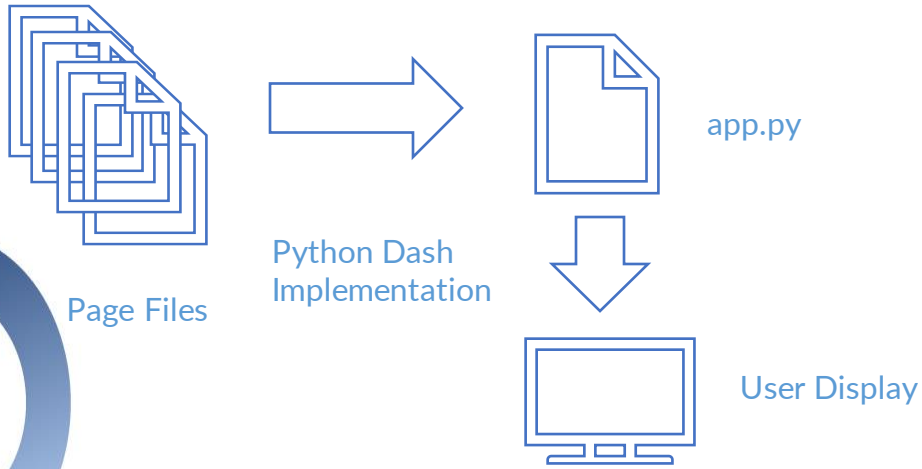
Python Dash -- One main file (app.py) with each tab its own executable file called by app.py

R Shiny – Two files, one for the user interface (ui.R) and one for the server and reactivity (server.R)



Comparisons – File Structure and Implementation

Python Dash	R Shiny
App file (can be named anything) handles interaction	ui.R and server.R interact to create a reactive environment
Call app file to run	Run app in directory with ui.R and server.R
One python file for each page	Tab layout all in ui.R
Calculations and data for each page in appropriate file	Calculations/data all in server.R
Reactivity available by callbacks using original functions	Reactivity attached to elements by wrapping output in specific pre-made functions

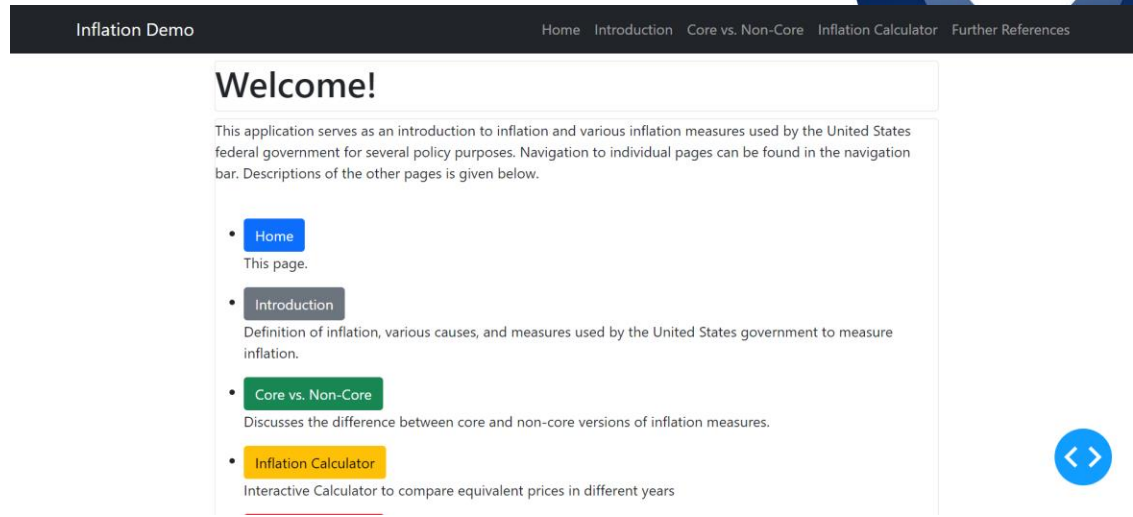


Home Page Implementation

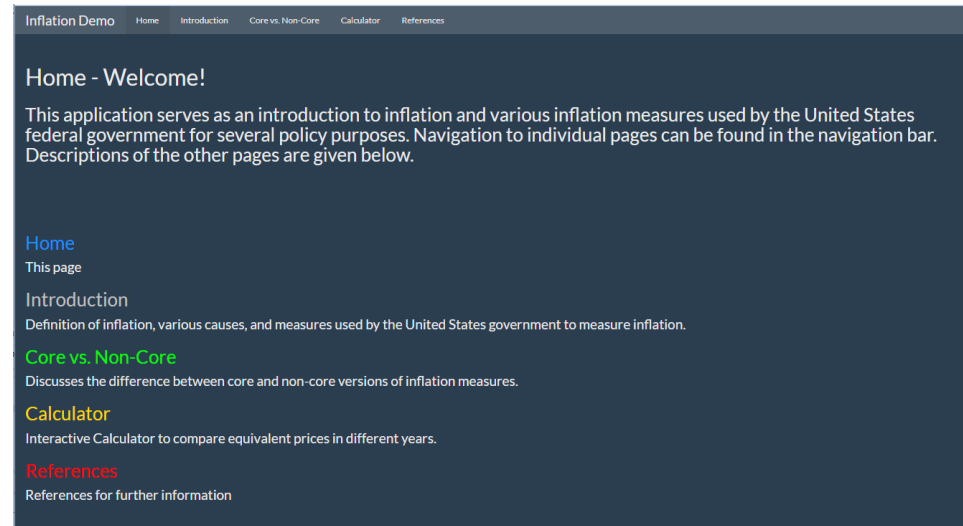
Python Dash

The home page in both apps serves as a link to both content pages. Python Dash gives a more modern web page look, particularly when using the Dash Bootstrap package.

R Shiny



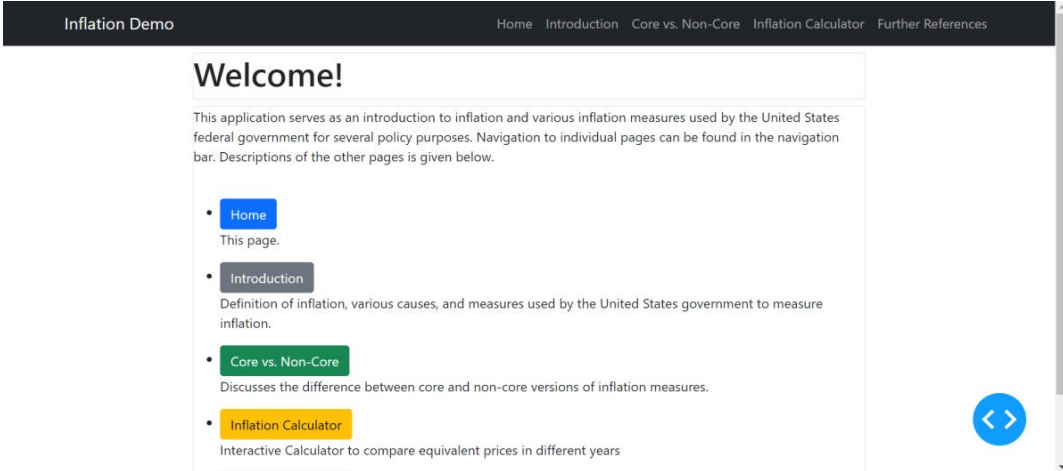
The screenshot shows the Python Dash application's home page. At the top, there is a dark navigation bar with the text "Inflation Demo" on the left and "Home Introduction Core vs. Non-Core Inflation Calculator Further References" on the right. The main content area has a white background with a "Welcome!" heading. Below the heading is a paragraph of introductory text. A list of four items follows, each with a colored button: "Home" (blue), "Introduction" (grey), "Core vs. Non-Core" (green), and "Inflation Calculator" (yellow). Each item has a short description. A blue double-arrow button is visible on the right side of the content area.



The screenshot shows the R Shiny application's home page. It has a dark grey background. At the top, there is a navigation bar with "Inflation Demo" on the left and "Home Introduction Core vs. Non-Core Calculator References" on the right. The main content area has a dark grey background with a "Home - Welcome!" heading. Below the heading is a paragraph of introductory text. A list of five items follows, each with a colored link: "Home" (blue), "Introduction" (grey), "Core vs. Non-Core" (green), "Calculator" (yellow), and "References" (red). Each item has a short description.



Home Page Implementation: Python Dash



```
import dash
from dash import html, dcc, callback, Input, Output
import dash_bootstrap_components as dbc

dash.register_page(__name__, path="/", title="Home", name="Home")

navbar = dbc.NavbarSimple(
    children = [dbc.NavItem(dbc.NavLink("Home", href="/")),
                dbc.NavItem(dbc.NavLink("Introduction", href="/inflation-intro")),
                dbc.NavItem(dbc.NavLink("Core vs. Non-Core", href="/core-inflation")),
                dbc.NavItem(dbc.NavLink("Inflation Calculator", href="/inflation-calculator")),
                dbc.NavItem(dbc.NavLink("Further References", href="/references"))],
    brand = "Inflation Demo",
    brand_href = "/",
    color = "dark",
    dark = True,
)

paragraph_1 = "This application serves as an introduction to inflation and various inflation " + \
"measures used by the United States federal government for several policy purposes. Navigation " + \
"to individual pages can be found in the navigation bar. Descriptions of the other " + \
"pages is given below."
```



```
main_card = dbc.Card(
    children = [
        html.P(paragraph_1),
        html.Br(),
        html.Ul(
            children = [
                html.Li([
                    dbc.Button("Home", href="/", color="primary", className="me-1"),
                    html.P(description_1),
                ]),
            ]
        )
    ]
)
```



```
body = html.Div(
    children = [
        dbc.Row(
            children = [
                dbc.Col(
                    children = [
                        title_card,
                    ],
                    width = {"size": 8, "offset": 2},
                ),
            ],
            style = {"margin": "8px"},
        ),
        dbc.Row(
            children = [
                dbc.Col(
                    children = [
                        main_card,
                    ],
                    width = {"size": 8, "offset": 2},
                ),
            ],
            style = {"margin": "8px"},
        ),
    ]
)
```



```
layout = html.Div(children=[
    navbar,
    body,
])
```



Home.py

App.py

```
import dash
from dash import html, dcc, callback, Input, Output
import dash_bootstrap_components as dbc

app = dash.Dash(
    __name__,
    use_pages=True,
    external_stylesheets=[dbc.themes.BOOTSTRAP]
)

app.layout = html.Div([
    dash.page_container
])

if __name__ == "__main__":
    app.run_server(debug=True)
```



Home Page Implementation: R Shiny

The screenshot shows a web application with a dark blue header containing navigation links: Inflation Demo, Home, Introduction, Core vs. Non-Core, Calculator, and References. The main content area has a dark blue background with white text. It starts with a heading 'Home - Welcome!' followed by a paragraph: 'This application serves as an introduction to inflation and various inflation measures used by the United States federal government for several policy purposes. Navigation to individual pages can be found in the navigation bar. Descriptions of the other pages are given below.' Below this are five links, each with a colored heading and a brief description: 'Home' (blue), 'Introduction' (white), 'Core vs. Non-Core' (green), 'Calculator' (yellow), and 'References' (red).



ui.R

```
#####
##### UI.R #####
#####
library(shiny)
library(shiny85)
library(shinythemes)
library(shinywidgets)
library(shinydashboard)

navbarPage("Inflation Demo",
  theme = shinytheme("superhero"),
  tabPanel("Home",
    h2("Home - Welcome!"),
    h3("This application serves as an introduction to inflation and various inflation measures used by the United States federal government for several policy purposes. Navigation to individual pages can
    br(),
    br(),
    br(),
    h3("Home", style = "color:dodgerblue"),
    h4("This page"),
    h3("Introduction", style = "color:silver"),
    h4("Definition of inflation, various causes, and measures used by the United States government to measure inflation."),
    h3("Core vs. Non-Core", style = "color:lime"),
    h4("Discusses the difference between core and non-core versions of inflation measures."),
    h3("Calculator", style = "color:gold"),
    h4("Interactive Calculator to compare equivalent prices in different years."),
    h3("References", style = "color:red"),
    h4("References for further information")
  )
),
```

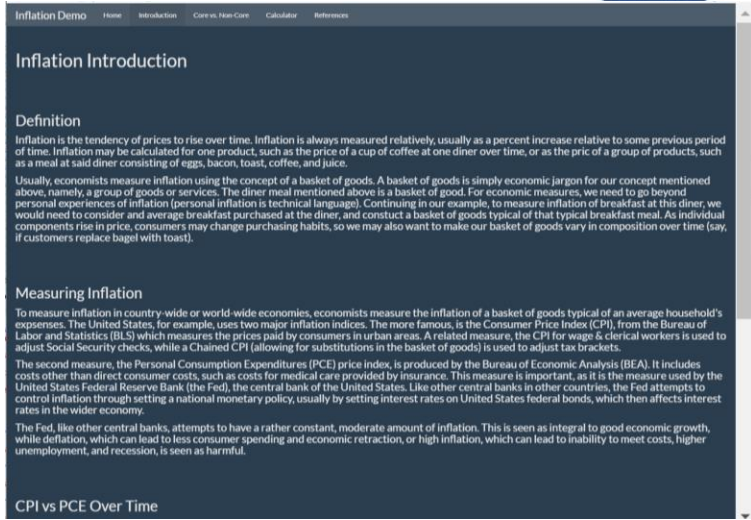
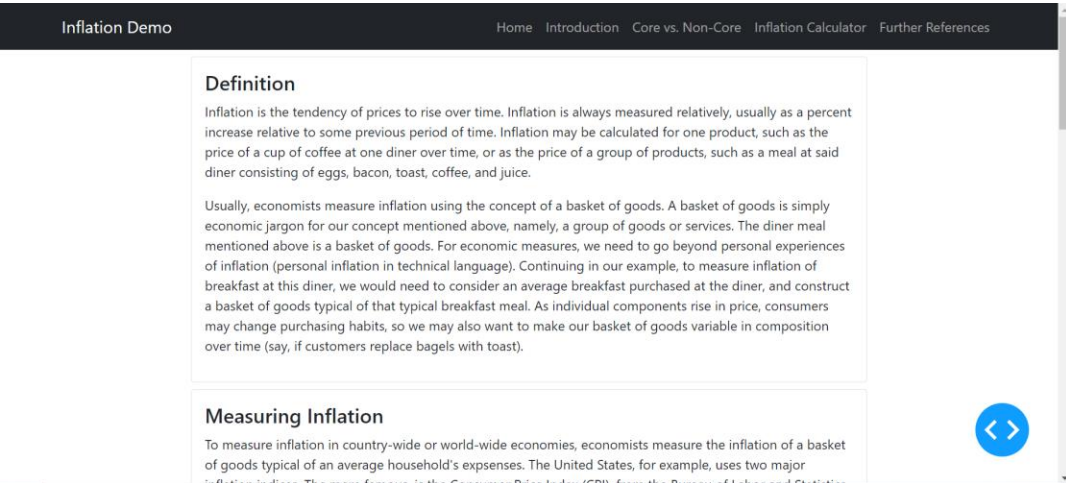


Project Controls
EXPO
Washington, DC - USA

Capabilities: Basic Web Page Layout

Python Dash

R Shiny



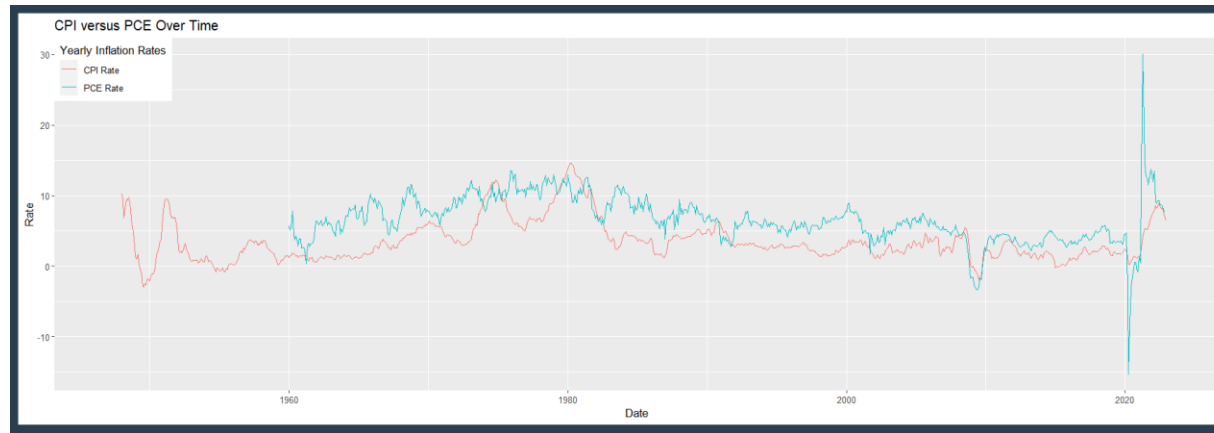
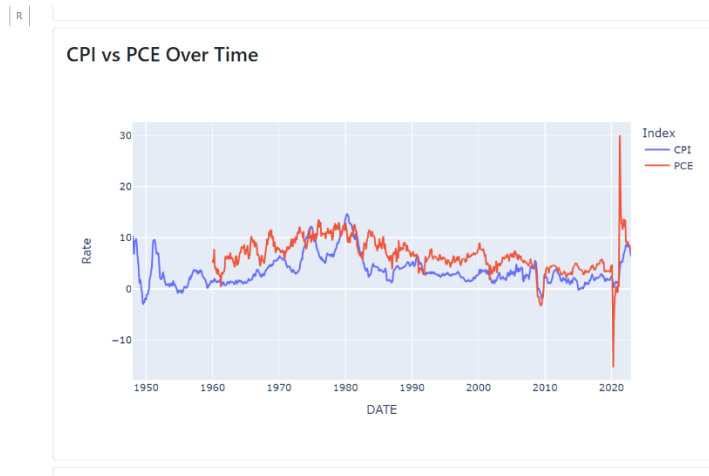
The Introduction and Core vs. Non-Core pages provide information on inflation definitions and measures, demonstrating basic layout, formatting, and graphics.

In Python Dash, the Dash Bootstrap package extends ease of formatting using a grid system.

Formatting, layout, and themes are more built-in in R Shiny. Themes in either case allow for consistent format of the page or pages.



Capabilities: Graphics



Python Dash

R Shiny

Capabilities: Graphics Implementation

Python Dash

```
def cpi_pce():
    # read data
    cpi_df = pd.read_csv("../data/CPI.csv")
    pce_df = pd.read_csv("../data/PCE_monthly.csv")

    # reset column names
    cpi_df.columns = ['DATE', 'CPI']
    pce_df.columns = ['DATE', 'PCE']

    # change data type
    cpi_df['DATE'] = pd.to_datetime(cpi_df['DATE'])
    pce_df['DATE'] = pd.to_datetime(pce_df['DATE'])

    # Calculate Rates
    cpi['CPI_Inflation_Rate'] = ((cpi['CPI'] - cpi['CPI_prev_yr'])/cpi['CPI_prev_yr'])*100
    pce['PCE_Inflation_Rate'] = ((pce['PCE'] - pce['PCE_prev_yr'])/pce['PCE_prev_yr'])*100

    # Merge Rates
    inflation = cpi.merge(pce, on = 'DATE', how = 'left')

    # Select rates only
    inflation = inflation.loc[:, ['DATE', 'CPI_Inflation_Rate', 'PCE_Inflation_Rate']]
    inflation.columns = ['DATE', 'CPI', 'PCE']

    # reshape data for graphing
    inflation = inflation.melt(id_vars = ['DATE'], value_name = 'Rate', var_name = 'Index')

    # return plotly figure for graphing
    fig = px.line(inflation, x = 'DATE', y = 'Rate', color = 'Index')

    return fig

card_graph = dbc.Card(
    dbc.CardBody(
        [
            html.H4("CPI vs PCE Over Time", className="card-title"),
            dcc.Graph(figure = cpi_pce()),
        ]
    ),
)
```

All code in inflation_intro.py. Layout code (bottom) calls function that returns a graph from the data (parts in top two pictures).

R Shiny

server.R

```
cpi_csv = "CPI.csv"
pce_csv = "PCE_monthly.csv"
pCore_csv = "PCE_core_monthly.csv"
cCore_csv = "CPI_core_seasonally_adjusted.csv"

shinyServer(function(input, output) {
  #Read in CPI and PCE monthly data
  cpi_df <- read.csv(cpi_csv)
  pce_df <- read.csv(pce_csv)
  pCore_df <- read.csv(pCore_csv)
  cCore_df <- read.csv(cCore_csv)

  #Set Column names, convert date from character string to date format
  names(cpi_df) <- c("DATE", "CPI")
  names(pce_df) <- c("DATE", "PCE")
  names(pCore_df) <- c("DATE", "Pcore")
  names(cCore_df) <- c("DATE", "Ccore")
  cpi_df$DATE<-as.Date(cpi_df$DATE,tryFormats = c("%Y-%m-%d"))
  pce_df$DATE<-as.Date(pce_df$DATE,tryFormats = c("%Y-%m-%d"))
  pCore_df$DATE<-as.Date(pCore_df$DATE,tryFormats = c("%Y-%m-%d"))
  cCore_df$DATE<-as.Date(cCore_df$DATE,tryFormats = c("%Y-%m-%d"))

  #create plot and send to UI
  output$CPIPCE <- renderPlot({
    p1<- ggplot(data=ratel, aes(x=DATE)) +
      geom_line(aes(y=CPIRate, colour = "blue")) +
      geom_line(aes(y=PCERate, colour = "red")) +
      labs(x="Date",y="Rate", title="CPI versus PCE Over Time",colour = "Yearly Inflation Rates") +
      theme(legend.position = c(0,1),legend.justification = c(0,1) +
        scale_colour_hue(labels = c("CPI Rate", "PCE Rate"))

    p1
  })
})
```

ui.R

```
h3("CPI vs PCE Over Time"),
plotOutput("CPIvPCE"),
br(),
```

Data load, processing, and graph rendering to ui element done in server.R. In ui.R, the plot element is declared.



Capabilities: Styling

Python Dash

Causes of Inflation

Inflation has a number of interlocking types, which are described below:

- Demand-pull inflation occurs when demand for goods and services exceeds the available supply. This can be due to increased consumer spending, government monetary policy, and/or (on the national level) increase in exports. This demand can somewhat cycle, as producers produce more and spend more on wages to increase supply. This is also the main cause attributed by laypeople.
- Cost-push inflation is the corresponding supply side decrease or increase in costs of raw materials or inputs. A subset of the most recent inflation is due to supply chain interruptions caused by the Russian full-scale invasion of Ukraine caused supply disruptions of Ukrainian food supplies and Russian energy supplies, which had effects on worldwide markets.
- On a national level, a devaluation of a country's currency can cause inflation. This is due to increased costs of imports, and a stimulation of local demand due to a shift in consumer behavior.
- As with any economic behavior, human behavior that affects the economy is due in part to human expectations of outcome. For example, increased wages in anticipation of inflation contributes to continued inflation.

Calculating Inflation Rates

To be technical, an inflation index is the price of a specified basket of goods at a specific period of time. This price might be normalized, so that one period of time is 1, and everything else is normalized to that time, such as inflation tables used in United States government appropriations. Commonly, inflation is presented as a rate, a percentage increase compared to some previous time. An inflation rate can be calculated:

$$\frac{\text{NewPrice} - \text{OldPrice}}{\text{OldPrice}} \cdot 100\%$$

Commonly, rates are a percent increase from a previous year, though the BLS, for instance, reports the CPI as increase over the previous month. In any case, to combine two rates, say, two years to get the total increase from year zero to year two, the rates need to be multiplied:

$$\text{TotalRate} = \left(\left(1 + \frac{\text{Rate}_1}{100\%} \right) \cdot \left(1 + \frac{\text{Rate}_2}{100\%} \right) - 1 \right) \cdot 100\%$$

R Shiny

Causes of Inflation

Inflation has a number of interlocking types, which are described below:

Demand-pull inflation occurs when demand for goods and services exceeds the available supply. This can be due to increased consumer spending, government monetary policy, and/or (on the national level) increase in exports. This demand can somewhat cycle, as producers produce more and spend more on wages to increase supply. This is also the main cause attributed by laypeople.

Cost-push inflation is the corresponding supply side decrease or increase in costs of raw materials or imports. Much of the most recent inflation is due to this type, due to supply chain interruptions caused by the COVID pandemic, after which demand outpaced supply. Additionally, the Russian full-scale invasion of Ukraine caused supply disruptions of Ukrainian food supplies and Russian energy supplies, which had effects on worldwide markets.

On a national level, a devaluation of a country's currency can cause inflation. This is due to increased costs of imports, and a stimulation of local demand due to a shift in consumer behavior.

As with any economic behavior, human behavior that affects the economy is due in part to human expectations of outcome. For example, increased wages in anticipation of inflation contributes to continued inflation.

Calculating Inflation Rates

To be technical, an inflation index is the price of a specified basket of goods at a specific period of time. This price might be normalized, so that one period of time is 1, and everything else is normalized to that time, such as inflation tables used in United States government appropriations. Commonly, inflation is presented as a rate, a percentage increase compared to some previous time. An inflation rate can be calculated:

$$\frac{\text{NewPrice} - \text{OldPrice}}{\text{OldPrice}} \cdot 100\%$$

Commonly, rates are a percent increase from a previous year, though the BLS, for instance, reports the CPI as increase over the previous month. In any case, to combine two rates, say, two years to get the total increase from year zero to year two, the rates need to be multiplied:

$$\text{TotalRate} = \left(\left(1 + \frac{\text{Rate}_1}{100\%} \right) \cdot \left(1 + \frac{\text{Rate}_2}{100\%} \right) - 1 \right) \cdot 100\%$$

The bottom of the Introduction page demonstrates styling capability. Both Python Dash and R Shiny have functions to produce many possible html formatting and styling tags.

Demonstrated here are bullet lists and markdown for equations. Other options include hyperlinks, video, images, and more.



Capabilities: Styling Implementation

Python Dash Bullet List

```

detail_exchange = "On a national level, a devaluation of a country's currency can cause inflation. " +
    "This is due to increased costs of imports, and a stimulation of local demand due to a shift in " +
    "consumer behavior."

detail_expectations = "As with any economic behavior, human behavior that affects the economy is " + \
    "due in part to human expectations of outcome. For example, increased wages in anticipation of " +
    "inflation contributes to continued inflation."

card3 = dbc.Card(
    dbc.CardBody(
        [
            html.H4("Causes of Inflation", className="card-title"),
            html.P(paragraph_causes),
            html.Ul([
                html.Li(detail_demand_pull),
                html.Li(detail_cost_push),
                html.Li(detail_exchange),
                html.Li(detail_expectations),
            ])
        ]
    ),
)

```

Python Dash Equation (Markdown)

```

paragraph_6 = "To be technical, an inflation index is the price of a specified basket of goods at a " +
    "specific period of time. This price might be normalized, so that one period of time is 1, and " +
    "everything else is normalized to that time, such as inflation tables used in United States " + \
    "government appropriations. Commonly, inflation is presented as a rate, a percentage increase " + \
    "compared to some previous time. An inflation rate can be calculated:"

paragraph_7 = "Commonly, rates are a percent increase from a previous year, though the BLS, for " + \
    "instance, reports the CPI as increase over the previous month. In any case, to combine two rates, " + \
    "say, two years to get the total increase from year zero to year two, the rates need to be multipli

equation_1 = dcc.Markdown("""
    
$$\frac{\text{New Price} - \text{Old Price}}{\text{Old Price}} \cdot 100\%$$

    """, mathjax = True)

equation_2 = dcc.Markdown("""
    Total Rate =  $\left(1 + \frac{\text{Rate}_1}{100}\right) \cdot \left(1 + \frac{\text{Rate}_2}{100}\right) - 1$ 
    """, mathjax = True)

card4 = dbc.Card(
    dbc.CardBody(
        [
            html.H4("Calculating Inflation Rates", className="card-title"),
            html.P(paragraph_6),
            equation_1,
            html.P(paragraph_7),
            equation_2,
        ]
    ),
)

```

R Studio Equation (Markdown)

```

h4("To be technical, an inflation index is the price of a specified basket of goods at a specific period of time. This price might be normalized, so that one period of time is 1, and everything else is normalized to t")
withMathJax(helpText("  $\frac{\text{New Price} - \text{Old Price}}{\text{Old Price}} \cdot 100\%$ "),
h4("Commonly, rates are a percent increase from a previous year, though the BLS, for instance, reports the CPI as increase over the previous month. In any case, to combine two rates, say, two years to get the total inc")
helpText(" Total Rate =  $\left(1 + \frac{\text{Rate}_1}{100}\right) \cdot \left(1 + \frac{\text{Rate}_2}{100}\right) - 1$ ")

```



Comparisons – Basic Functionality

Category	Python Dash	R Shiny
Result	Outputs HTML from programming language	Outputs HTML from programming language
File Structure	Runs an app.py file	Needs ui.R (for ui) and server.R (for reactivity)
Page Implementation	Pages folder with a .py file for each page (can have tabs on a page)	Essentially runs multiple pages as tabs
Layout, Formatting, Themes	Can build on grid system more easily with Dash Bootstrap	Grid system built into layout components; Themes already selectable
Styling	General styling (text, lists, markdown for equations)	General styling (text, lists, markdown for equations)
Graphics	Good with inbuilt Plotly (better than Matplotlib)	Ggplot2 is better than Base R (probably worse)
Miscellaneous	Can script in IDE, run on command line; Slower to load with multiple pages; Callback fails in runtime	Can develop/run in RStudio IDE; Sometimes error causes an annoying crash



Comparisons – Reactive Programming

Inflation Calculator Introduction

The calculator below calculates equivalent dollar value from one month to another, using four inflation measures (CPI, PCE, Core CPI, and Core PCE).

Calculation is possible for the timeframe that data exists for all four measures, namely 1959-2022.

To begin, enter a dollar value in the input box.

Next, choose the month to convert from, then the month to convert to.

To calculate, press the button marked 'Calculate'.

Input Amount

Enter a dollar amount to convert from below:

Start Month



The Inflation Calculator escalates a dollar amount from one month to another using four different inflation measures. The above image is a gif playable in Slide Show mode.

Comparisons – Reactive Programming Implementation

Python Dash

Data Read and Input Handling

```

# Read the CSV files into pandas DataFrames
cpi_core = pd.read_csv("../data/cpi_core_monthly_adjusted.csv")
pce_core = pd.read_csv("../data/pce_core_monthly.csv")
pce_core = pd.read_csv("../data/pce_core_monthly.csv")

# Create columns and change first column datatype to a datetime object
cpi_core.columns = ['date', 'core_cpi']
pce_core.columns = ['date', 'core_pce']
pce_core.columns = ['date', 'core_pce']

cpi_core['date'] = pd.to_datetime(cpi_core['date'])
pce_core['date'] = pd.to_datetime(pce_core['date'])
pce_core['date'] = pd.to_datetime(pce_core['date'])

# Function to run inflation calculation and generate output
def get_old_values(in_month, in_year, out_month, out_year, input_dollars):

    # get dates as datetime objects
    in_date = datetime.datetime.strptime(in_year, '%Y')
    out_date = datetime.datetime.strptime(out_year, '%Y')

    # get values
    old_cpi = cpi_core.loc[(cpi_core['date'] == in_date) & (cpi_core['date'] == out_date)]
    old_pce = pce_core.loc[(pce_core['date'] == in_date) & (pce_core['date'] == out_date)]
    old_core_cpi = cpi_core.loc[(cpi_core['date'] == in_date) & (cpi_core['date'] == out_date)]
    old_core_pce = pce_core.loc[(pce_core['date'] == in_date) & (pce_core['date'] == out_date)]
    
```

User Interface Layout

```

button_card = dbc.Card(
    dbc.CardBody(
        [
            html.H4("Calculate", className="card-title"),
            html.P("Press the button to calculate inflation:"),
            dbc.Button(
                "calculate", id="calculate-button", className="me-2", n_clicks=0
            ),
            html.Div(id="output-message", children = [])
        ]
    ),
)

```

```

@callback(
    output("output-message", "children"),
    input("calculate-button", "n_clicks"),
    State("input-dollars", "value"),
    State("input-month", "value"),
    State("input-year", "value"),
    State("output-month", "value"),
    State("output-year", "value"),
)
def get_output_message(n_clicks, input_dollars, input_month, input_year, output_month, output_year):
    # return nothing if not clicked or a value is missing
    if n_clicks == 0:
        return []
    # return message
    else:
        # get data/frame
        df = get_old_values(int(input_month), int(input_year), int(output_month),
                           int(output_year), input_dollars)

        # construct output components
        components = [html.P(f"{row[1]}") for row in df.iterrows()]

        return components

```

Layout and User Interface

Callback to Render Calculator Output

In Python Dash, when the button is pushed, the callback runs a function to display the output, calling another function that reads the input and handles the data.

In R Shiny, the interface is defined in ui.R, with the output mapped and rendered by server.R. One difference from the graph, is that a button is used to only display output if clicked.

R Shiny ui.R

```

tabPanel("Calculator",
  N3("Inflation Calculator"),
  N3("Inflation Calculator Introduction"),
  N4("The calculator below calculates equivalent dollar value from one month to another, using four inflation measures (CPI, PCE, Core CPI, and Core PCE)."),
  N4("Calculation is possible for the timeframe that data exists for all four measures, namely 1959-2022."),
  N4("To begin, enter a dollar value in the input box."),
  N4("Next, choose the month to convert from, then the month to convert to."),
  N4("To calculate, press the button marked 'Calculate'.",
  inflation_calculator:
    br(),
    N3("Input Amount"),
    N4("Enter a dollar amount to convert from below"),
    numericInput("inputdollars", label = "", value = 0),
    br(),
    N3("Start Month"),
    N4("Select the month to convert from:"),
    selectInput("inputMonth", label = "Month", choices = c(" " = 0, "January" = 1, "February" = 2, "March" = 3, "April" = 4, "May" = 5, "June" = 6, "July" = 7, "August" = 8, "September" = 9, "October" = 10, "November" = 11, "December" = 12)),
    N3("End Month"),
    N4("Select the month to convert to:"),
    selectInput("outputMonth", label = "Month", choices = c(" " = 0, "January" = 1, "February" = 2, "March" = 3, "April" = 4, "May" = 5, "June" = 6, "July" = 7, "August" = 8, "September" = 9, "October" = 10, "November" = 11, "December" = 12)),
    selectInput("outputYear", label = "Year", choices = c(" " = 0, "1959-2022")),
    N3("Calculate"),
    N4("Press the button to calculate inflation!"),
    actionButton("getbutton", "Calculate", class = "btn-primary"),
    HTMLOutput("calculatorOutput")
),
)

```

server.R

```

observeEvent(input$getbutton, {
  dollars = input$inputdollars
  startMonth = input$inputMonth
  startYear = input$inputYear
  endYear = input$outputYear

  fromCPI = get_dollars(cpi_df[DATE] == startYear & month(cpi_df[DATE]) == startMonth, 'CPI')
  toCPI = get_dollars(cpi_df[DATE] == endYear & month(cpi_df[DATE]) == endMonth, 'CPI')
  fromPCE = get_dollars(pce_df[DATE] == startYear & month(pce_df[DATE]) == startMonth, 'PCE')
  toPCE = get_dollars(pce_df[DATE] == endYear & month(pce_df[DATE]) == endMonth, 'PCE')
  fromCoreCPI = get_dollars(core_cpi_df[DATE] == startYear & month(core_cpi_df[DATE]) == startMonth, 'Core')
  toCoreCPI = get_dollars(core_cpi_df[DATE] == endYear & month(core_cpi_df[DATE]) == endMonth, 'Core')
  fromCorePCE = get_dollars(core_pce_df[DATE] == startYear & month(core_pce_df[DATE]) == startMonth, 'Core')
  toCorePCE = get_dollars(core_pce_df[DATE] == endYear & month(core_pce_df[DATE]) == endMonth, 'Core')

  newDollarsCPI = round(dollars * toCPI/fromCPI, 2)
  newDollarsPCE = round(dollars * toPCE/fromPCE, 2)
  newDollarsCoreCPI = round(dollars * toCoreCPI/fromCoreCPI, 2)
  newDollarsCorePCE = round(dollars * toCorePCE/fromCorePCE, 2)
  dollarsBounded = round(dollars, 2)

  startMonthString = paste(month(as.numeric(startMonth), label = TRUE, abbr = FALSE))
  endMonthString = paste(month(as.numeric(endMonth), label = TRUE, abbr = FALSE))

  outputStringCPI = paste("By the CPI index, $", dollarsBounded, " in ", startMonthString,
    ", ", startYear, " is equivalent to $", newDollarsCPI, " in ",
    endMonthString, ", ", endYear, ". sep = """)
  outputStringPCE = paste("By the PCE index, $", dollarsBounded, " in ", startMonthString,
    ", ", startYear, " is equivalent to $", newDollarsPCE, " in ",
    endMonthString, ", ", endYear, ". sep = """)
  outputStringCoreCPI = paste("By the Core CPI index, $", dollarsBounded, " in ", startMonthString,
    ", ", startYear, " is equivalent to $", newDollarsCoreCPI, " in ",
    endMonthString, ", ", endYear, ". sep = """)
  outputStringCorePCE = paste("By the Core PCE index, $", dollarsBounded, " in ", startMonthString,
    ", ", startYear, " is equivalent to $", newDollarsCorePCE, " in ",
    endMonthString, ", ", endYear, ". sep = """)

  outputString = paste(outputStringCPI, outputStringPCE, outputStringCoreCPI, outputStringCorePCE, sep = "\n")
  output$calculatorOutput <- renderUI(HTML(outputString))
})

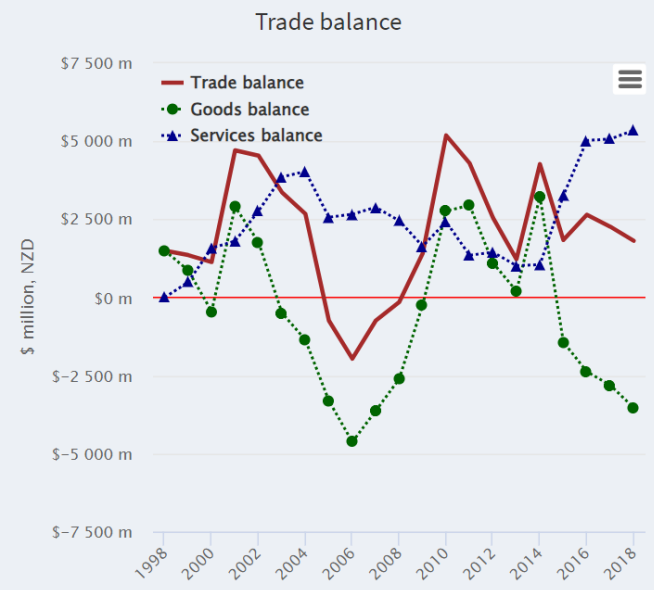
```

Calculator Output Rendering



Aside: A Really Reactive Open Source Example

New Zealand trade over the past 20 years



<https://gallery.shinyapps.io/nz-trade-dash/>



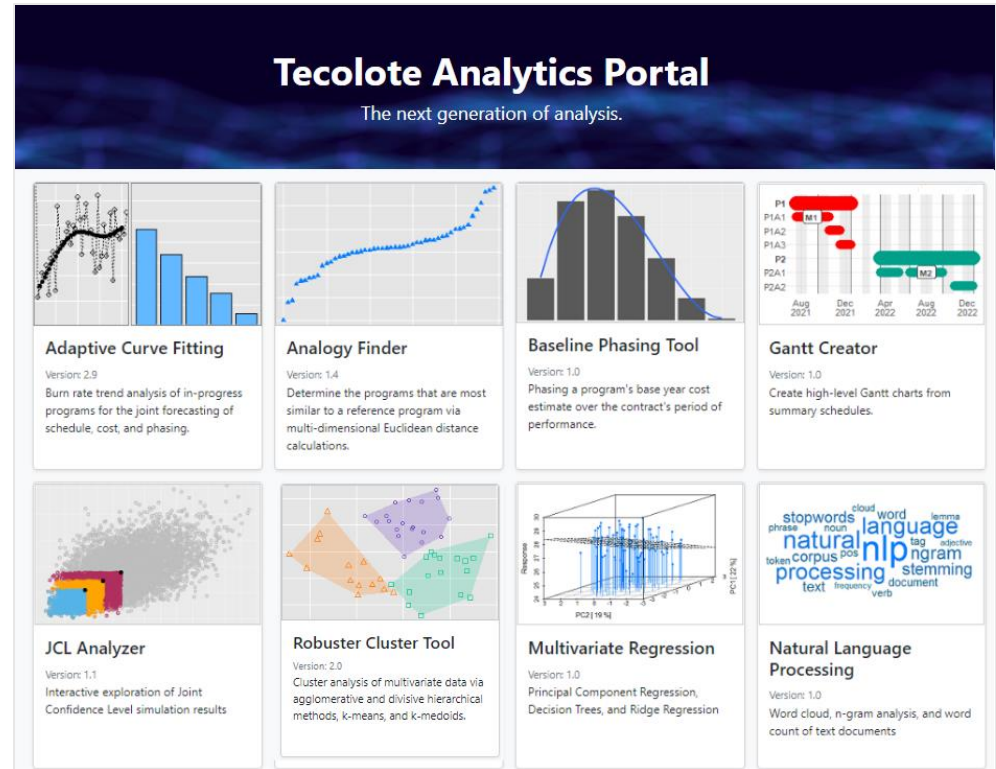
Comparisons – Implementing on an Enterprise Level

- Commercial cloud-hosting options exist for both languages/packages
 - Dash Design Kit from Plotly (for Dash)
 - Posit Connect from Posit (for Shiny)
- Some open source options also exist, such as the free ShinyServer for R and Shiny
- Internal hosting is also an option (e.g., Tecolote Analytics Portal)
- Organizational choice will depend on available blend of resources:
 - Technical
 - Human
 - Financial
 - Security



Internal hosting Tecolote Analytics Portal (TAP)

- An internal (intranet only) web portal that hosts Web Applications such as R-Shiny and Python-Dash web apps.
- These applications are developed by analysts across the company for use by a broader audience



Conclusions

- Shiny in R and Dash in Python can both be used for similar web application functionality
 - Shiny is good for clean dashboards and loading data applicable for multiple tabs at once
 - Dash and Python is good for modularity and real web page/dashboard development
- Differences in some capability/file structure
 - Dash allows more complex file structure, which can be better organized, but may have loading speed implications
 - More extensive styling functionality available with additional packages in Dash that is somewhat inherent to Shiny
 - Plotly comes with Dash and has inbuilt high-level graphics
- R and Python's inherent language design creates differences
 - R is created for statistics, so has many inbuilt statistical functions
 - Python is designed as a programming language
 - Capabilities accessed by additional packages
 - Graphics capability arguably more advanced
- Best option will depend on user experience, IT capability, enterprise hosting options, etc.



Further Discussion and Future Thoughts

- Web applications are great as live dashboards
 - Professionally transitioning to a more virtual society
 - Pull live data from various sources
 - Online tools for modeling
 - Reactive programming
- Web applications do require effort to develop and possibly maintain
 - The more reactive the program or complex the styling, the longer this is
 - Version updates may be important
 - Enterprise-level hosting and security are challenging
- In some instances, BI tools (i.e. Tableau, Power BI, etc.) may be more advantageous
- Web forms are more complex than dashboards



Acknowledgements

- Pablo Barajas, Dr. Michael Schiavoni, Dr. Daniel Newkirk
- Tecolote Research, Inc.
- ICEAA
- You, the viewers



References

- Some introductions to inflation:
- [Investopedia](#)
- [International Monetary Fund](#)
- [European Central Bank](#)
- Some further information on causes of inflation:
- [Reserve Bank of Australia](#)
- Additionally, information on various inflation measurements:
- [Brookings](#)
- [Investopedia](#)
- [San Francisco Federal Reserve](#)
- The historical United States inflation data was sourced from:
- [Federal Reserve Economic Data](#)



THANK YOU

Inflation Measures Review



Inflation is the increase of prices over time

Government bodies use inflation measures to track inflation

These are typically a basket of goods (a group of items, often typically used by a household)

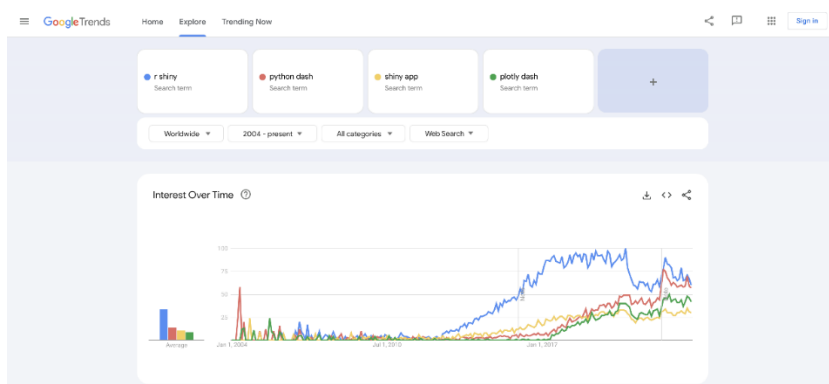
The Consumer Price Index (CPI) from the BLS is used to adjust tax brackets and social security payments

The Personal Consumption Expenditures (PCE) index is used by the Fed to adjust interest rates

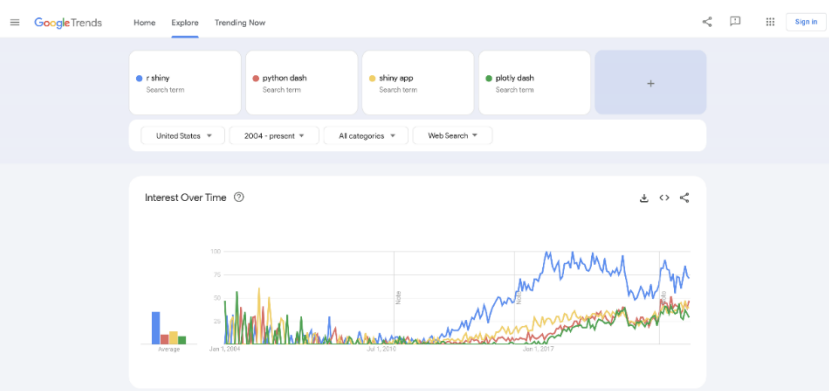
Core inflation does not include highly variable food and fuel prices, and is useful to track alongside total inflation



Comparisons – Community and General Considerations



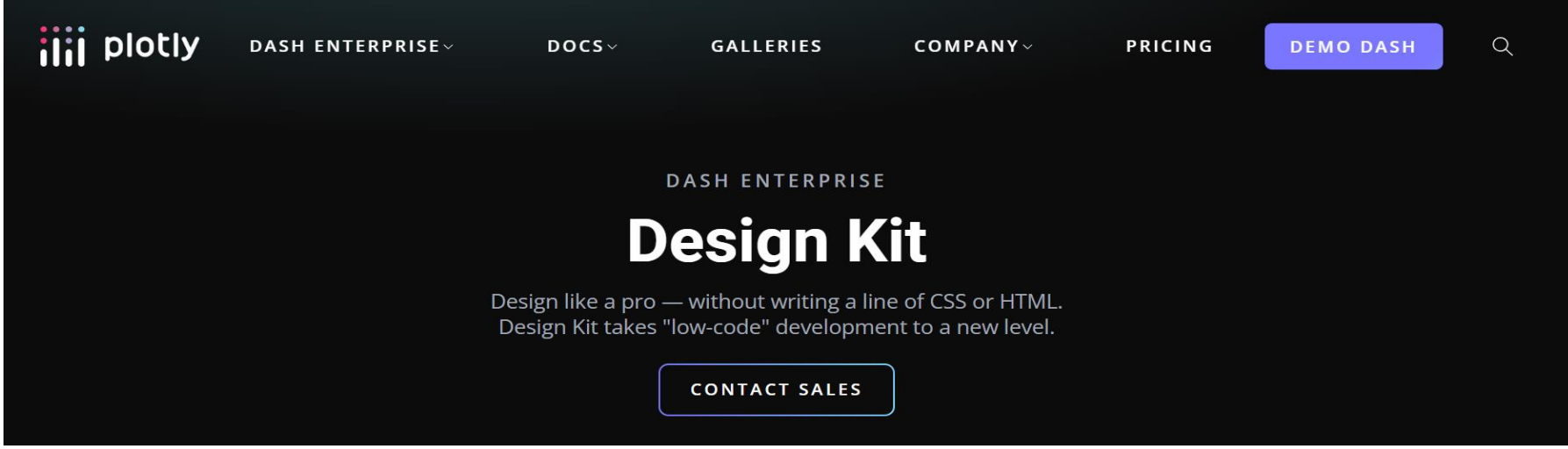
Worldwide Search Trends



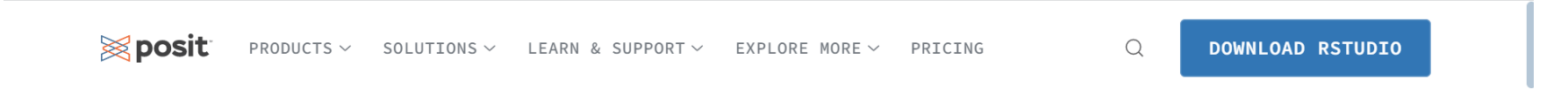
United States Search Trends

R Shiny has more historical usage, though Python Dash is catching up (More worldwide than in the United States)

Comparisons – Implementing on an Enterprise Level



The screenshot shows the Plotly website's 'Design Kit' page. The navigation bar includes 'plotly', 'DASH ENTERPRISE', 'DOCS', 'GALLERIES', 'COMPANY', 'PRICING', and a 'DEMO DASH' button. The main heading is 'DASH ENTERPRISE Design Kit'. Below the heading, a sub-heading reads 'Design like a pro — without writing a line of CSS or HTML. Design Kit takes "low-code" development to a new level.' A 'CONTACT SALES' button is centered below the text.



The screenshot shows the Posit website's 'SHINY SERVER' page. The navigation bar includes 'posit', 'PRODUCTS', 'SOLUTIONS', 'LEARN & SUPPORT', 'EXPLORE MORE', 'PRICING', a search icon, and a 'DOWNLOAD RSTUDIO' button. The main heading is 'SHINY SERVER'.

SHINY SERVER

Get your Shiny apps online

