Algorithm and Programming Final Project Report

Machine Learning Exploration

Name: Matthew Staniswinata
Class: L1AC
Student ID: 2702325962

Computer Science

Course Code: COMP6047001
Course Name: Algorithm and Programming
Class: L1AC
Lecturer: Jude Joseph Lamug Martinez,MCS
Type of Assignment: Final Project report

Bina Nusantara International University
Jakarta, 12 January 2024

A. Introduction

This is the project that I have been working on for quite some time. Here this is a web-based project, where users can explore the idea of machine learning. I got the inspiration working on this project, because I myself have been in a position of trying to learn machine learning and artificial intelligence, but don't have much motivation for something that will drive me to learn more. Because I don't see many things that I believe I can do about machine learning.

I'm decent at mathematics and never really tried coding before when I was younger. But something about AI and machine learning did pique my interest, as modern technologies are all automated. Even engineers and developers could put a machine and make them learn something. One of the examples would be the chess AI engine from Deep Blue that was able to beat at that time chess champion Garry Kasparov. And other big examples like automated cars and weather predictions, etc.

I strive myself to learn, but due to many of those AIs that I have seen on the internet. They are pretty much out of my league to even understand what's going on. When I'm in that position, I'm trying to learn machine learning by doing. Such as working on projects and following tutorials but with still inconsistency.

So, here comes my project idea as it might help with other people who are interested in machine learning, but just find them too complicated and hard to start with. Here my project strives to be used as motivation and inspiration for the newcomers. This project will be about machine learning, and you can apply this idea as part of data scientists as well. Because users will be knowing about some machine learning models, widely known such as Linear Regression, K-Nearest Neighbours, Random forest and Decision tree algorithm that is being used as a regression algorithm.

Users are able to experience hands-on about what machine learning does with such as prediction and regression algorithms and peak their interests about looking at relationships of dataset.

B. Project Specification/Description:

This program is based on the Dash from Plotly module which is going to be the website integration development of this project.

Modules/Tools:

- Dash - Website and Interface development
- Scikit learn - Creating the algorithm models

- Plotly - Creating the graph/diagram interfaces
- Pandas - Read and process data
- Numpy - Creating intervals for the data
- import base64 - Encoding of the provided dataset
- import io - Read the csv file from pandas dataframe, and creating stream of string
- Datetime - To tell the date/time of the processed data
- Scipy - web visualization and style of the diagram

The inputs needed for this program are:

- Dataset: Custom ones. Preferably dataset coming from official resources that already have pre-confirmation
- Model choice: User able to choose what type of algorithm that they want to see

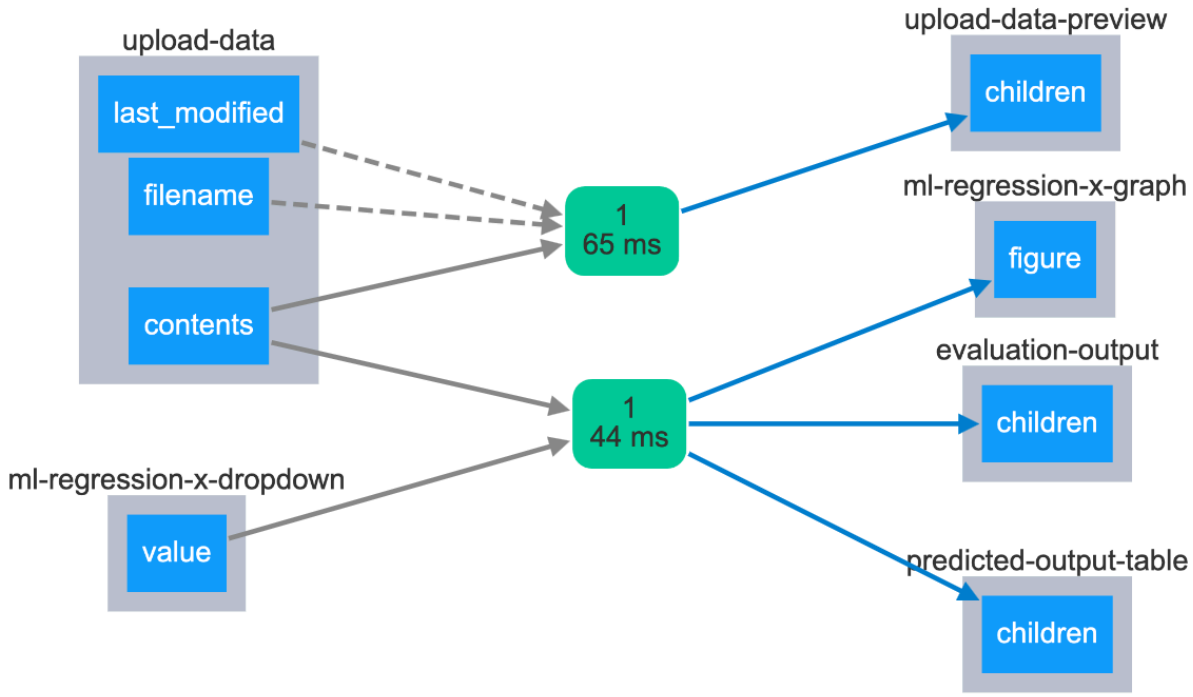
The outputs from this program are:

- Display of the uploaded data in table format
- Visual diagram, looking at the relationship between the actual and predicted data
- Evaluation output, to look at the error of the relationship and their correlations
- Display of all points for the actual and predicted data in table format

C. Solution design:

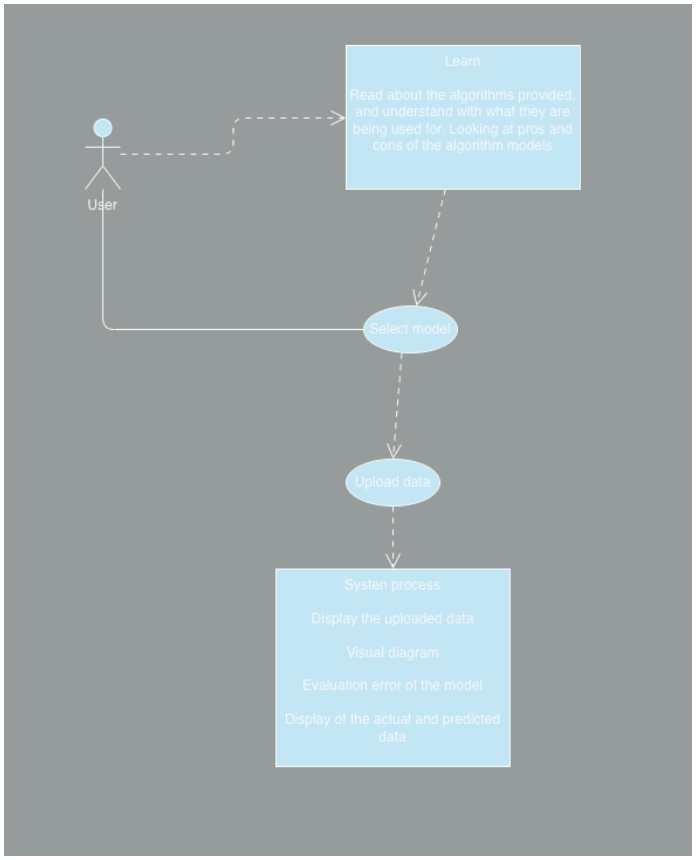
For my solution design, I created it in a website for the main reason of being accessible to everyone without any hindrance coming from them to learn. As knowledge is everywhere and for anyone.
The design of the website doesn't have many interfaces or designs that catch people's eyes when they find it. But this might be the first step for users to start getting used to when they are dividing into the world of learning. As I have found myself through experience, when I started learning about machine learning there are mathematics involved and to get into the zone of learning. People should start by reading texts, even though it's boring it will become effective as there won't be any distractions and people will start to grow their attention span to be longer and more effective and having the patience mentally in learning.
Interface of my design tries to follow the research paper with texts, images, and additional of the program inside of the website. Simple old classic style format of a website, as many websites that we used to learn more about mathematical and in-depth theory of technology information they will be something like this, simple and not much going on but straight-forward.

- use case diagram, activity diagram, class diagram:

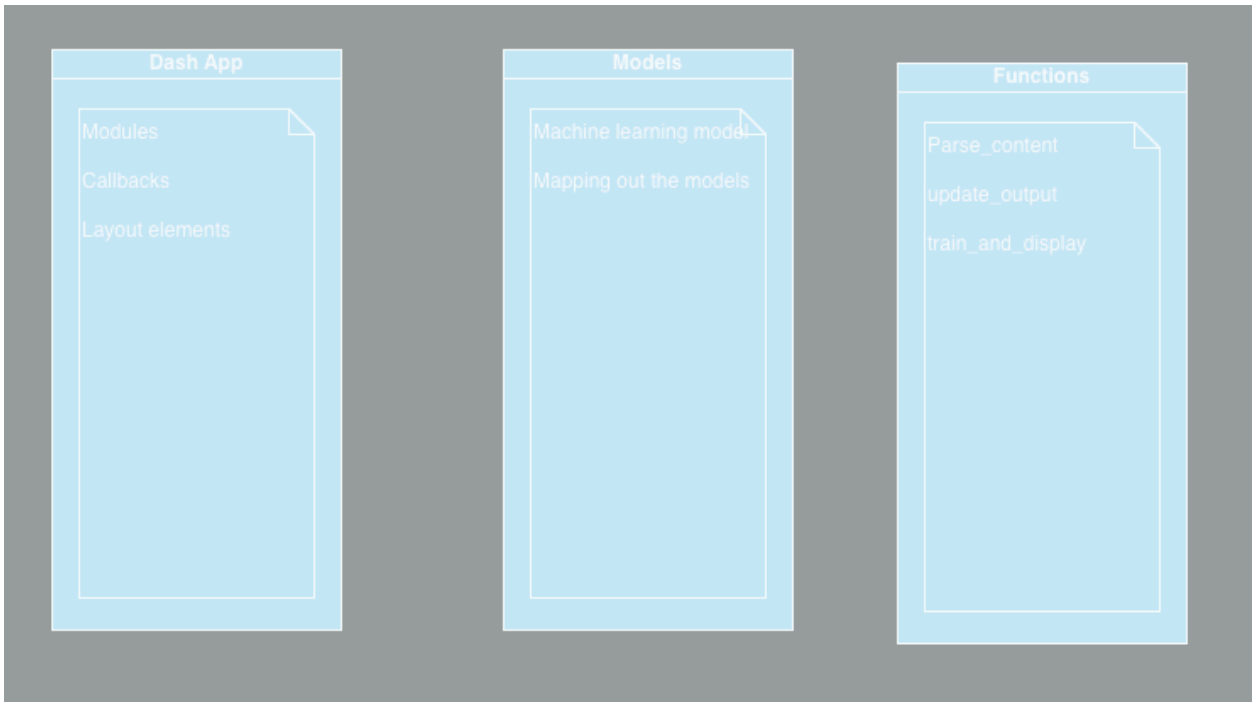


User - Process - Website
The green shape consider as the process time of the website working

Upload-data: they will be getting the information of the last_modified, filename, and contents of the file
Ml-regression-x-dropdown: Choosing the machine learning model
Ml-regression-x-graph: Displays a graphical representation of the machine learning model's performance
Upload-data-preview: It says children, as there are many previews and formats edited
Evaluation-output: Same with upload-data-preview, as there are many additional formats but used for output for the data points used into the table output and graph output
Predicted-output-table: Format of the table for the evaluation output



Because this is a website, users can go back or refresh to restart or do anything back and forth.



D. Explanations of the code:

Application layout is defined using HTML and Dash components, which include:

1. Descriptions of machine learning models - as from the welcoming words at the top
2. A dropdown menu for selecting models - 4 options of the algorithm models
3. A file upload component - Where it will be essential input to run the provided program
4. Components for displaying the graph, evaluation metrics, and predictions - Output for users

```
from dash import Dash, dcc, html, dash_table, Input, Output, State, callback
from sklearn.model_selection import train_test_split
from sklearn import linear_model, tree, neighbors
import plotly.graph_objects as go
import pandas as pd
import numpy as np
import base64
import io
import datetime
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scipy.stats import pearsonr
from dash.exceptions import PreventUpdate
from dash.dash_table.Format import Format, Group, Scheme, Symbol
```

Import Statements:
dash for creating the web application, scikit learn(sklearn) for implementing the machine learning models, plotly for generating the plots, pandas and numpy for handling data, base64 and io for handling file uploads, datetime for timestamping, and scipy for calculating Pearson correlation coefficient.

```
# Set up dash application
app = Dash(__name__)
```

This is used as the main setup for running the Dash web application

```
models = {
    'Regression': linear_model.LinearRegression,
    'Decision Tree': tree.DecisionTreeRegressor,
    'k-NN': neighbors.KNeighborsRegressor,
    'Random Forest': RandomForestRegressor
}
```

Model dictionary: All 4 models that I will be using. Calling out the id name:model import

```
# Linear Regression
linear_regression_section = html.Div([
   html.H2('Linear Regression'),
   html.Img(src=app.get_asset_url('linear_regression.png'), style={'width': '20%',
'height': 'auto', 'objectFit': 'contain'}),
   html.P('Linear regression is a statistical method that allows us to study
relationships between two continuous (quantitative) variables.'
          ' One variable is considered to be an explanatory variable, and the other is
considered to be a dependent variable.'),
   html.P('Pros: Simple to implement and interpret, works well with small datasets.'),
   html.P('Cons: Assumes a linear relationship between variables, sensitive to
outliers, assumes constant variance of residuals.'),
])

# Decision Tree
decision_tree_section = html.Div([
   html.H2('Decision Tree'),
   html.Img(src=app.get_asset_url('decision_tree.png'), style={'width': '20%',
'height': 'auto', 'objectFit': 'contain'}),
   html.P('A decision tree is a flowchart-like structure in which each internal node
represents feature(or attribute), each leaf node represents class label (decision
taken after evaluating all features), and each branch represents a rule.'),
   html.P('Pros: Easy to understand and interpret, handles both categorical and
numerical data, less prone to overfitting.'),
   html.P('Cons: Can create biased trees if some classes dominate, sensitive to small
changes in data.'),
])

# Random Forest
random_forest_section = html.Div([
   html.H2('Random Forest'),
   html.Img(src=app.get_asset_url('random_forest.png'), style={'width': '20%',
'height': 'auto', 'objectFit': 'contain'}),
   html.P('Random forest is a meta estimator that fits a number of decision tree
classifiers on various sub-samples of the dataset and uses averaging to improve the
predictive accuracy and control over-fitting.'),
   html.P('Pros: Works well with both categorical and numerical data, less likely to
overfit than single decision trees, provides feature importance.'),
   html.P('Cons: Less interpretable than decision trees, computationally expensive
with large datasets.'),
```

```
])

# K-NN
knn_section = html.Div([
   html.H2('K-NN'),
   html.Img(src=app.get_asset_url('KNN.png'), style={'width': '20%', 'height': 'auto',
'objectFit': 'contain'}),
   html.P('K-NN is a simple, easy-to-implement supervised machine learning algorithm
that can be used to solve both classification and regression problems.'),
   html.P('Pros: Simple to understand and implement, works well with multi-class
classification, less prone to overfitting.'),
   html.P('Cons: Computationally intensive with large datasets, sensitive to
irrelevant attributes.'),
])
```

Algorithm's descriptions where it shows information about the algorithm
Here we have html components coming from Dash as web applications.

```
app.layout = html.Div([
   html.H1('Welcome to Machine Learning Exploration', style={'textAlign': 'center'}),
   html.H3('Where you can start learning and getting your interests high to start your
machine learning journey. With our playground that covers some basics about machine
learning and what do they do', style={'textAlign': 'center'}),
   html.Div(linear_regression_section, style={'textAlign': 'center'}),
   html.Div(decision_tree_section, style={'textAlign': 'center'}),
   html.Div(random_forest_section, style={'textAlign': 'center'}),
   html.Div(knn_section, style={'textAlign': 'center'}),
   dcc.Location(id='url', refresh=False),
   html.Div(id='page-content'),
   html.A(html.Button("Check out the code in my github"),
          href="https://github.com/MS-092/Algo_FProject"),

   # Choosing model algorithms
   html.H2(
       'Now here I have provided you some model algorithms playground for you to
play'),
   html.H3('Here you can choose any provided model algorithms, with your own custom
dataset'),
   html.P("Select model:"),
   dcc.Dropdown(
       id='ml-regression-x-dropdown',
       options=["Regression", "Decision Tree", "k-NN", "Random Forest"],
```

```
        value='Decision Tree',
        clearable=False
    ),
    # Upload dataset - Parsing upload files
    dcc.Upload(
        id='upload-data',
        children=html.Div([
            'Drag and Drop or ',  # drag and drop area
            html.A('Select Files')  # upload button
        ]),
        style={
            'width': '100%',
            'height': '60px',
            'lineHeight': '60px',
            'borderWidth': '1px',
            'borderStyle': 'dashed',
            'borderRadius': '5px',
            'textAlign': 'center',
            'margin': '10px'
        },
        multiple=False
    ),
    # Set up the layout
    html.Div(id='upload-data-preview'),
    dcc.Graph(id="ml-regression-x-graph"),
    dcc.Markdown(id='evaluation-output'),
    dcc.Markdown(id='predicted-output'),
    html.Div(id='predicted-output-table')
])
```

This section defines the overall layout of the Dash application. It includes a welcome message, descriptions of the available models, a dropdown menu for selecting a model, an upload button for uploading a dataset, and areas for displaying the uploaded data, the model's performance, evaluation metrics, and prediction results.

```
# Parse upload files, getting the information of the data
# Based on the type of file being uploaded
def parse_contents(contents, filename, date):
    # This check ensures that there is a comma in the contents to split
    if ',' in contents:
        content_type, content_string = contents.split(',')
    else:
        # Handle the case where contents does not contain a comma
        return html.Div([
            'There was an error processing this file.'
        ])

    decoded = base64.b64decode(content_string)
    try:
        if 'csv' in filename:
            # Assume that the user uploaded a CSV file
            df = pd.read_csv(io.StringIO(decoded.decode('utf-8')))
        elif 'xls' in filename:
            # Assume that the user uploaded an excel file
            df = pd.read_excel(io.BytesIO(decoded))
    except Exception as e:
        print(e)
        return html.Div([
            'There was an error processing this file.'
        ])

    return html.Div([
        html.H5(filename),
        html.H6(datetime.datetime.fromtimestamp(date)),
        dash_table.DataTable(
            df.to_dict('records'),
            [{'name': i, 'id': i} for i in df.columns],
            page_action='native',
            page_size=10
        ),
        html.Hr(),
        html.Div('Raw Data'),
```

```
        html.Pre(contents[0:10] + '...', style={
            'whiteSpace': 'pre-wrap',
            'wordBreak': 'break-all'
        })
    ])
```

File Parsing Function: The parse_contents function takes the contents of an uploaded file along with its name and last modified date. It decodes the contents and reads them into a DataFrame, which is then displayed using a DataTable component. This function is used to give users a preview of their uploaded data.

```
@app.callback(
    Output('upload-data-preview', 'children'),
    Input('upload-data', 'contents'),
    State('upload-data', 'filename'),
    State('upload-data', 'last_modified')
)
def update_output(list_of_contents, list_of_names, list_of_dates):
    if list_of_contents is not None:
        # Make sure all variables are lists to handle single or multiple file uploads
        if not isinstance(list_of_contents, list):
            list_of_contents = [list_of_contents]
        if not isinstance(list_of_names, list):
            list_of_names = [list_of_names]
        if not isinstance(list_of_dates, list):
            list_of_dates = [list_of_dates]

        children = [
            parse_contents(c, n, d) for c, n, d in
            zip(list_of_contents, list_of_names, list_of_dates)
        ]
        return children
```

Data upload and Preview Callback: function update_output is responsible for upload-data-preview in the output component of the app. So, whenever a user uploads a file it will take the content of the file, name, and date of the file as information formality as the input. We can see it from the Input and State components of the app callback.
The function ensures that these inputs are listed to handle multiple file uploads and uses the parse_contents function to generate previews of the uploaded files.

```
@app.callback(
    Output("ml-regression-x-graph", "figure"),
    Output('evaluation-output', 'children'),
    Output('predicted-output-table', 'children'),  # Add this line for the table output
```

```
    Input('ml-regression-x-dropdown', "value"),
    Input('upload-data', 'contents'),
)
def train_and_display(name, contents):
    if contents is None:
        # If there is no data uploaded
        raise PreventUpdate
    else:
      content_type, content_string = contents.split(',')
      decoded = base64.b64decode(content_string)
      df = pd.read_csv(io.StringIO(decoded.decode('utf-8')))

      # Fetch the first two column names
      feature_col = df.columns[0]
      target_col = df.columns[1]

      # Dataframe is being split as part of the function
      X = df[feature_col].values[:, None]
      X_train, X_test, y_train, y_test = train_test_split(
          X, df[target_col], random_state=42)

      model = models[name]()
      model.fit(X_train, y_train)

      predictions = model.predict(X_test)

      x_range = np.linspace(X.min(), X.max(), 100)
      y_range = model.predict(x_range.reshape(-1, 1))

      fig = go.Figure([
          go.Scatter(x=X_test.squeeze(), y=y_test,
                     name='test', mode='markers'),
          go.Scatter(x=x_range, y=y_range,
                     name='prediction')
      ])

      mse = mean_squared_error(y_test, predictions)
      mae = mean_absolute_error(y_test, predictions)
      r2 = r2_score(y_test, predictions)
      corr, _ = pearsonr(y_test, predictions)

      eval_output = f"""
```

```
 **Model:** {name} \n
 **Mean Squared Error:** {mse} \n
 **Mean Absolute Error:** {mae} \n
 **R-squared:** {r2 * -1} \n
 **Correlation Coefficient:** {corr}
 """
     # average absolute difference between actual and predicted values - the lower the
better value or going into 0%
     # how well regression model explained observed values/data - 0 until 1

     # Showing the list of the actual data and predicted output dataset
     # Convert the actual and predicted values to a DataFrame
     pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': predictions})

     # Create a DataTable to display the actual and predicted values
     table = dash_table.DataTable(
       data=pred_df.to_dict('records'),
       columns=[
           {'name': 'Actual', 'id': 'Actual', 'type': 'numeric', 'format':
Format(precision=2, scheme=Scheme.fixed)},
           {'name': 'Predicted', 'id': 'Predicted', 'type': 'numeric', 'format':
Format(precision=2, scheme=Scheme.fixed)}
       ],
       sort_action='native', # Enable sorting system in the table output
       style_table={'width': '50%'},  # Adjust the width of the table as needed
       style_cell={'textAlign': 'center'}
   )

     return fig, eval_output, table # Return statement and bringing out all of the
output
```

Model training part: The train_and_display function is triggered when a user selects a model from the dropdown menu and uploads a dataset. This function processes the uploaded data, trains the selected model, and generates predictions. It then creates a Plotly graph to visualize the predictions against the test data, calculates evaluation metrics (MSE, MAE, R-squared, and Pearson correlation), and displays them. Additionally, it generates a DataTable to show the actual and predicted values side by side.

Display callback:
Outputs:

1. The "ml-regression-x-graph" output updates the figure in the graph component with the ID ml-regression-x-graph.

2. The 'evaluation-output' output updates the children of a Markdown component where the evaluation metrics will be displayed.
3. The 'predicted-output-table' output updates the children of a component where the predictions table will be displayed.

Inputs:

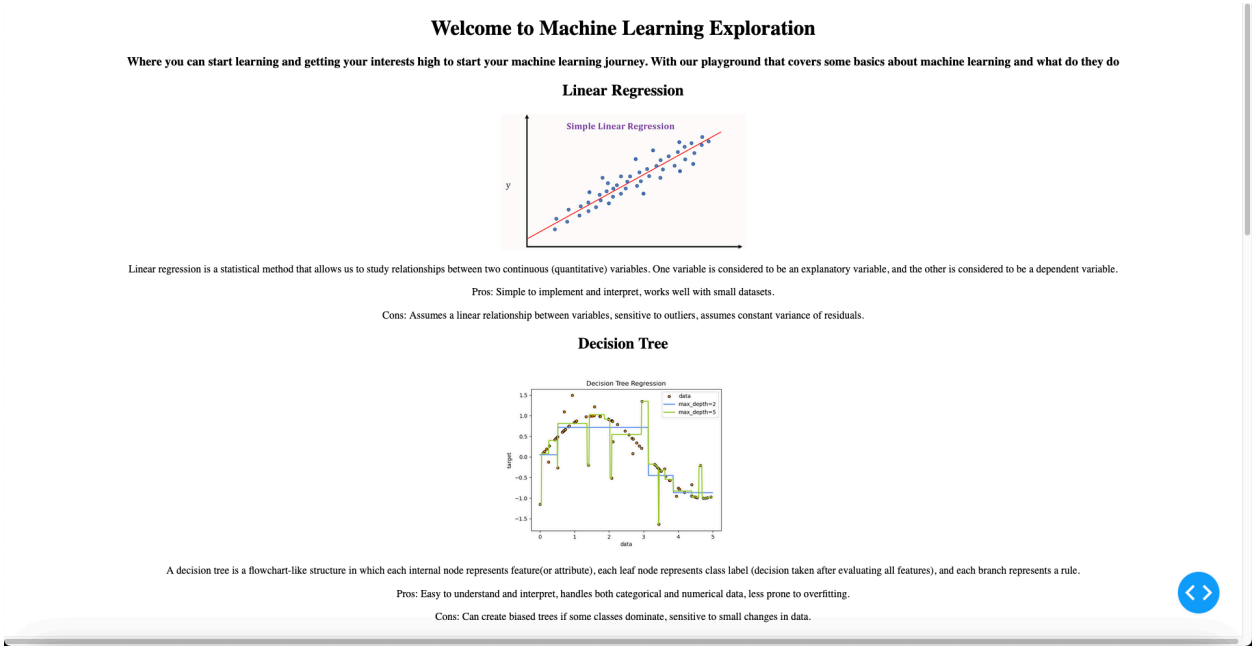
1. The "ml-regression-x-dropdown" input listens for changes in the value of the dropdown component with the ID ml-regression-x-dropdown, which corresponds to the selection of a machine learning model.
2. The 'upload-data' input listens for changes in the contents of the upload-data component, which is used to upload the dataset for training and testing.

Overall these callbacks are functions that define the behavior of the application in response to user interactions. They handle events such as uploading a file, selecting a model, training the selected model on the uploaded data, making predictions with the trained model, and updating the graph and evaluation metrics.

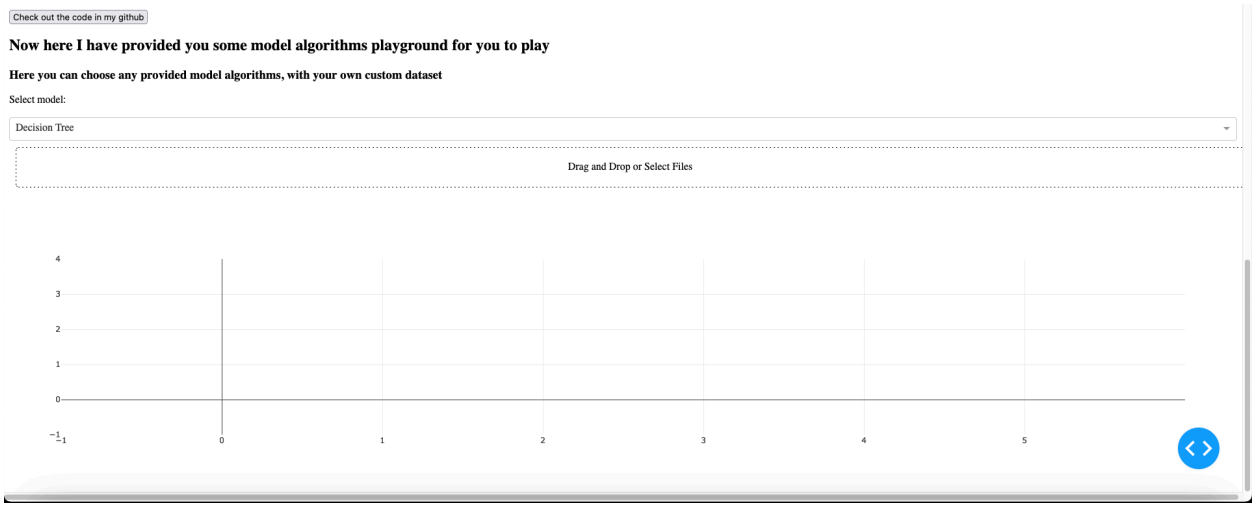
```
# Main function to run the whole program process
if __name__ == "__main__":
   app.run_server(debug=True)
```

Main function to run the whole code process

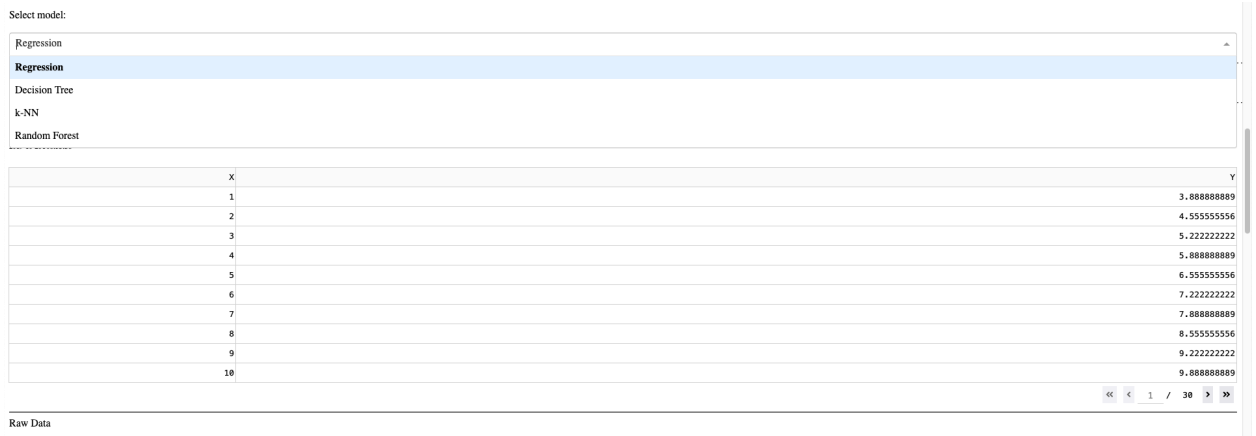
E. Proof or ways to show that the program works:



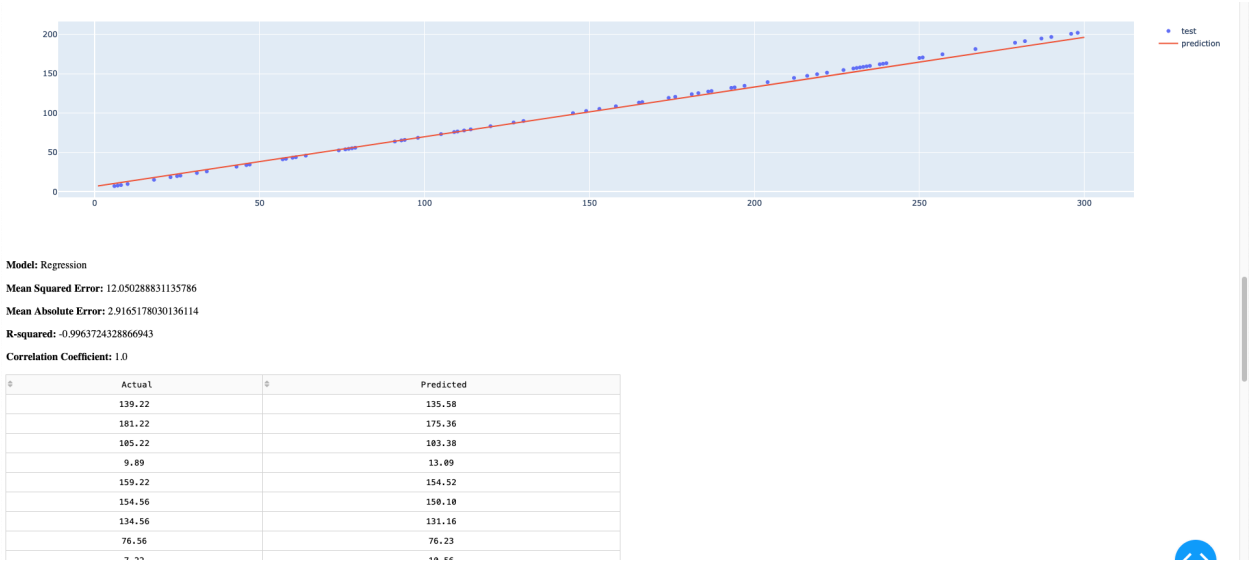
Beginning of the page, showing an image of each algorithm in a diagram and what they might look like but still based on the scale or dataset itself.
Providing the algorithm's name and brief description, with pros and cons on using this model and why you should or should not use this algorithm model.



Here, I put the link button to the github repository.
Model dropdown is there and can be changed.
Diagram display ready when the data is being uploaded
Users upload the data by selecting files from the computer, or drag-and-dropping the file.



Here I have uploaded the data, and when it's uploaded. Users can see a display of the uploaded data.
Users are also able to choose any of those 4 algorithm models.



Here is the final output that user sees:
Visual diagram
Evaluation error of the model with the data
Display data points between the actual and predicted data points

F. Video demo:
https://drive.google.com/file/d/1J2GQgKrbViKnUxm35hH2eANFpS2iHCFd/view?usp=sharing

G. References:

Documentations:
Plotly. (n.d.). Dash: Python framework for building analytical web applications. Retrieved January 12, 2024, from https://dash.plotly.com
Scikit-Learn. (n.d.). Machine Learning in Python. Retrieved January 12, 2024, from https://scikit-learn.org/stable/

Tutorials/Inspirations:

Charming Data. (2020). *Introduction to Dash Plotly - Data Visualization in Python . YouTube*. Retrieved January 12, 2024, from https://www.youtube.com/watch?v=hSPmj7mK6ng.

Lianne and Justin. (2021a). *Python Interactive Dashboards with Plotly Dash - Quick Tutorial . YouTube*. Retrieved January 12, 2024, from https://www.youtube.com/watch?v=UYGwgHhazMA.

Real Python. (2021b). *Starting Data Visualizations With Dash and Python . YouTube*. Retrieved January 12, 2024, from https://www.youtube.com/watch?v=pGMvvq7R1IM.

Asset images:
Scikit-Learn. (n.d.). Decision tree. Retrieved January 12, 2024, from https://scikit-learn.org/stable/modules/tree.html

Stanford University. (n.d.). K-nearest neighbors. Retrieved January 12, 2024, from https://web.stanford.edu/class/stats202/notes/Linear-regression/K-nearest-neighbors.html

Data-Driven Investor. (n.d.). Machine Learning 101: Part 1. Retrieved January 12, 2024, from https://medium.datadriveninvestor.com/machine-learning-101-part-1-24835333d38a

Samet Girgin. (2019, April 7). Random Forest Regression in 5 Steps with Python. Medium. Retrieved January 12, 2024, from https://medium.com/@sametgirgin/random-forest-regression-in-5-steps-with-python-ee4259eca0de

H. Student's declaration

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of student: