INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 7, Issue 2 - V7I2-1309)
Available online at: https://www.ijariit.com

Note booker pro case study

Mainak Chaudhuri
mainakc24365@gmail.com
SRM Institute of Science and Technology, Chennai, Tamil Nadu

ABSTRACT

The main idea of the software named, "Notebooker Pro" is to make the task of preparing a data science notebook, super easy and fast. With this software, any person can quickly get the insights about a data set displayed in front of him. This software also provides effective visualization facilities with 6 types of commonly used graphs. Anyone can also compare through 30 different Machine Learning models of both regression and classification types to choose the best model that can provide the highest accuracy for a data set of a given size and split ratio, (which can also be set by the user as per his/her requirements).

***Keywords*:** *Data Analysis, Machine Learning, Data Visualization, Model Building, Web Application, Data Science Notebooks, Education, Speed of Development.*

1. INTRODUCTION

Data Science has been a developing stream of technical knowledge, since the beginning of 2nd decade of the 21st century. Many companies, public and private sectors, governing bodies and startups are investing big deals in this field to help themselves analyse the ever increasing volumes of data generated by their customers every moment. Several tools have been developed to handle big data, but yet there was not enough light thrown on the learning path. An average person takes 3 to 6 whole months to get a complete knowledge about data science with all its basics. But, it becomes a tedious task for a learner or teacher of this discipline to make a good data science notebook, generally made as a project to understand and teach data science. They need to type the lengthy lines of code each and every time for a different notebook to get the simplest jobs done, like displaying useful insights about data. Sometimes, having no prior plans or pathway for a project, people tend to loose interest and leave tasks midway, only to never return and solve the problem. This leads to delay in project delivery and not many students and teachers can succeed in making a proper and channelized notebook in the first attempt.

2. PRIMARY ISSUES WITH CONVENTIONAL APPROACH

1. **Manual planning**: For making a good notebook, the creator needs to plan it well. The notebook must be informative and include all necessary details yet not over-burdening it with a flood of information which are irrelevant.

2. **Time taken to code**: Once the planning is done, now comes the most tedious and brain-storming part, that is to code the notebook into existence. This can take hours, and also result in aches in various body parts and a bit of monotony, working for the same thing for so long.

3. **Finding a good accuracy**: It is definitely a hard job to find the appropriate train-test-split ratio in order to increase the accuracy of the model. At times, due to a poor size selection, the test set and the train set data do not agree with each other, resulting in high training accuracy but low-test set accuracy.

4. **Messing up with syntax to make a proper graph**: Making a good graph with proper axes/parameters is also a great challenge for a beginner or a person with a limited time frame.

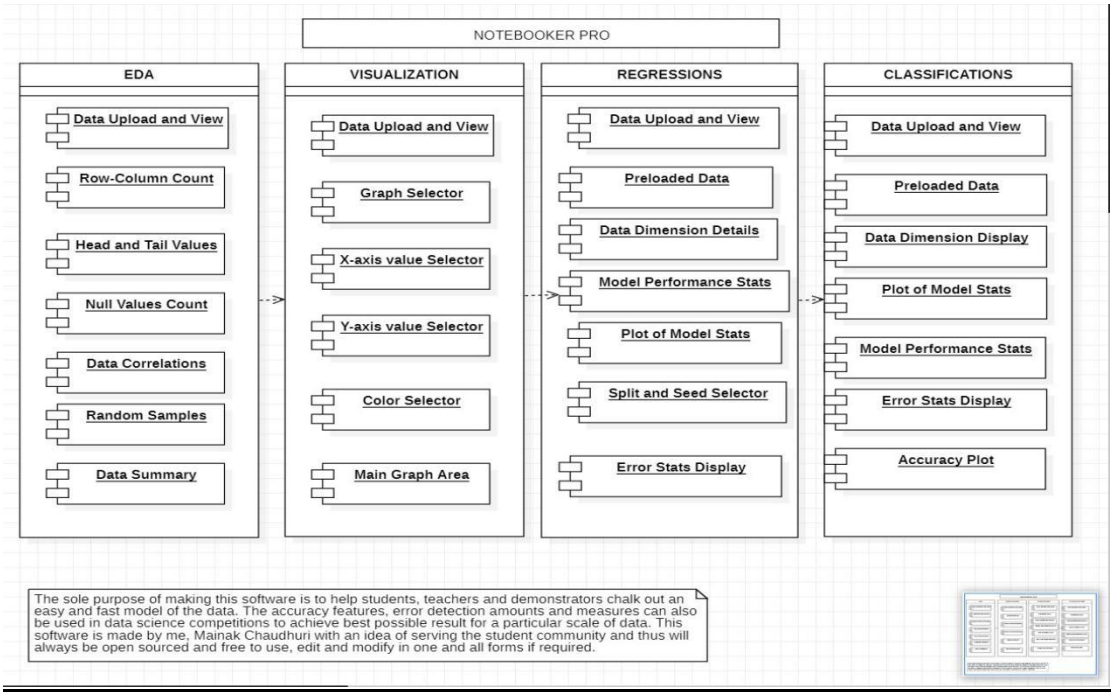
3. METHODOLOGY

The approach to solve the above-mentioned problems was through a object oriented modelling of the problem statement and then solving these models individually, only to be put into one after the entire project (Divide and Conquer). There are 5 important models to take care of. They are:
i. Data import or upload.

ii. Extraction of insights.
iii. Generating graphs from the data available.
iv. Generate multiple regression models from the data and then comparing the best model.
v. Generating classification models from the data and then comparing the best model for a given split ratio and data randomization constant.

In the software, all the above-mentioned steps can be performed independently as per the need and feel of a process. So, the usage is made as simple as possible.

4. UML DIAGRAM



5. CODE FOR THE PROJECT

```
import streamlit as st
import numpy as np
import pandas as pd
import plotly_express as px
from sklearn import *
from lazypredict.Supervised import LazyRegressor
from lazypredict.Supervised import LazyClassifier
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import base64
import io
import webbrowser
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes
from PIL import Image


image = Image.open('ds.jpg')
st.image(image,use_column_width=True)

def main():
    activities = ['EDA','Visualization','Regression','Classification','Documentation','About Us']
    #st.sidebar.success('Updates Coming Soon!')
    option=st.sidebar.selectbox('Choose a section',activities)
    st.sidebar.markdown('''Use this section for finding useful insights about your data,and feel free to use them in your notebooks


Version : 1.0.2  ''')
```



```
    if option == 'EDA':
        st.subheader("Explanatory Data Analysis")

        data=st.file_uploader("Please upload a CSV dataset ",type=['csv'])



        st.warning('Your dataset goes here...')
        if data is not None:
            df=pd.read_csv(data)
            st.dataframe(df)
            st.info('Some useful data insights about your data')
            if st.checkbox("Display shape"):
                r,c = df.shape
                st.write('Rows = ',r,'Columns = ',c)

            if st.checkbox('Display columns'):
                st.write(df.columns)

            if st.checkbox('Select multiple columns'):
                selected_col = st.multiselect('Select preferred columns',df.columns)
                df1 = df[selected_col]
                st.dataframe(df1)

            if st.checkbox("Head"):
                st.write(df.head())

            if st.checkbox('Tail'):
                st.write(df.tail())

            if st.checkbox('Null values'):
                st.write(df.isnull().sum())

            if st.checkbox('Data types'):
                st.write(df.dtypes)

            if st.checkbox('Random sample'):
                st.write(df.sample(20))

            if st.checkbox('Display correlations'):
                st.write(df.corr())

            if st.checkbox('Summary'):
                st.write(df.describe(include='all').T)

    elif option == 'Visualization':
        st.subheader("Data Visualization and Graphing")

        st.sidebar.subheader("File Upload")

        # Setup file upload
        uploaded_file = st.sidebar.file_uploader(
                            label="Upload your CSV file. (200MB max)",
                            type=['csv'])


        if uploaded_file is not None:
            st.success('Your data goes here')


        try:
            df = pd.read_csv(uploaded_file)
        except Exception as e:
            st.warning('Data not found')
```



```
global numeric_columns
global non_numeric_columns
try:
   st.write(df)
   numeric_columns = list(df.select_dtypes(['float', 'int']).columns)
   non_numeric_columns = list(df.select_dtypes(['object']).columns)
   non_numeric_columns.append(None)
   print(non_numeric_columns)
except Exception as e:
   print(e)



chart_select = st.sidebar.selectbox(
   label="Select the chart type",
   options=['Scatterplots', 'Lineplots', 'Histogram', 'Boxplot','Violinplot','Piechart']
)

st.info('The Graphs generated will be displayed here')

if chart_select == 'Scatterplots':
   st.sidebar.subheader("Scatterplot Settings")
   try:
      x_values = st.sidebar.selectbox('X axis', options=numeric_columns)
      y_values = st.sidebar.selectbox('Y axis', options=numeric_columns)
      color_value = st.sidebar.selectbox("Color", options=non_numeric_columns)
      plot = px.scatter(data_frame=df, x=x_values, y=y_values, color=color_value)
      # display the chart
      st.plotly_chart(plot)
   except Exception as e:
      print(e)

if chart_select == 'Lineplots':
   st.sidebar.subheader("Line Plot Settings")
   try:
      x_values = st.sidebar.selectbox('X axis', options=numeric_columns)
      y_values = st.sidebar.selectbox('Y axis', options=numeric_columns)
      color_value = st.sidebar.selectbox("Color", options=non_numeric_columns)
      plot = px.line(data_frame=df, x=x_values, y=y_values, color=color_value)
      st.plotly_chart(plot)
   except Exception as e:
      print(e)

if chart_select == 'Histogram':
   st.sidebar.subheader("Histogram Settings")
   try:
      x = st.sidebar.selectbox('Feature', options=numeric_columns)
      bin_size = st.sidebar.slider("Number of Bins", min_value=10,
                                   max_value=100, value=40)
      color_value = st.sidebar.selectbox("Color", options=non_numeric_columns)
      plot = px.histogram(x=x, data_frame=df, color=color_value)
      st.plotly_chart(plot)
   except Exception as e:
      print(e)

if chart_select == 'Boxplot':
   st.sidebar.subheader("Boxplot Settings")
   try:
      y = st.sidebar.selectbox("Y axis", options=numeric_columns)
      x = st.sidebar.selectbox("X axis", options=non_numeric_columns)
      color_value = st.sidebar.selectbox("Color", options=non_numeric_columns)
      plot = px.box(data_frame=df, y=y, x=x, color=color_value)
      st.plotly_chart(plot)
   except Exception as e:
      print(e)
```



```
    if chart_select == 'Piechart':
        st.sidebar.subheader("Piechart Settings")
        try:
            x_values = st.sidebar.selectbox('X axis', options=numeric_columns)
            y_values = st.sidebar.selectbox('Y axis', options=non_numeric_columns)
            plot = px.pie(data_frame=df, values=x_values, names=y_values)
            st.plotly_chart(plot)

        except Exception as e:
            print(e)

    if chart_select == 'Violinplot':
        st.sidebar.subheader("Violin Plot Settings")
        try:
            x_values = st.sidebar.selectbox('X axis', options=numeric_columns)
            y_values = st.sidebar.selectbox('Y axis', options=numeric_columns)
            color_value = st.sidebar.selectbox("Color", options=non_numeric_columns)
            plot = px.violin(data_frame=df, x=x_values, y=y_values, color=color_value)
            st.plotly_chart(plot)
        except Exception as e:
            print(e)

elif option == 'Regression':
    st.subheader("Regression ML Model Builder")

    # Model building
    def build_model(df):
        l = len(df)

        #df = df.iloc[:100]
        X = df.iloc[:,:-1] # Using all column except for the last column as X
        Y = df.iloc[:,-1] # Selecting the last column as Y


        st.markdown('**1.2. Dataset dimension**')
        st.write('X (Independent Axis)')
        st.info(X.shape)
        st.write('Y (Dependent Axis)')
        st.info(Y.shape)

        st.markdown('**1.3. Variable details**:')
        st.write('X variable (first few are shown)')
        st.info(list(X.columns[:int(l/5)]))
        st.write('Y variable')
        st.info(Y.name)

        # Build lazy model
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y,test_size = split_size,random_state = seed_number)
        reg = LazyRegressor(verbose=0,ignore_warnings=False, custom_metric=None)
        models_train,predictions_train = reg.fit(X_train, X_train, Y_train, Y_train)
        models_test,predictions_test = reg.fit(X_train, X_test, Y_train, Y_test)

        st.subheader('2.Model Performance Plot (Training Set)')

        st.write('Training set')
        st.write(predictions_train)
        st.markdown(filedownload(predictions_train,'training.csv'), unsafe_allow_html=True)

        st.write('Test set')
        st.write(predictions_test)
        st.markdown(filedownload(predictions_test,'test.csv'), unsafe_allow_html=True)

        st.subheader('3.Model Performance Plot(Test set)')
```



```
    with st.markdown('**R-squared**'):
        # Tall
        predictions_test["R-Squared"] = [0 if i < 0 else i for i in predictions_test["R-Squared"] ]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="darkgrid")
        ax1 = sns.barplot(y=predictions_test.index, x="R-Squared", data=predictions_test)
        ax1.set(xlim=(0, 1))
    st.markdown(imagedownload(plt,'plot-r2-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(12, 3))
    sns.set_theme(style="darkgrid")
    ax1 = sns.barplot(x=predictions_test.index, y="R-Squared", data=predictions_test)
    ax1.set(ylim=(0, 1))
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-r2-wide.pdf'), unsafe_allow_html=True)

    with st.markdown('**RMSE (capped at l/2)**'):
        # Tall
        predictions_test["RMSE"] = [(l/2) if i > (l/2) else i for i in predictions_test["RMSE"] ]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="darkgrid")
        ax2 = sns.barplot(y=predictions_test.index, x="RMSE", data=predictions_test)
    st.markdown(imagedownload(plt,'plot-rmse-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(12, 3))
    sns.set_theme(style="darkgrid")
    ax2 = sns.barplot(x=predictions_test.index, y="RMSE", data=predictions_test)
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-rmse-wide.pdf'), unsafe_allow_html=True)

    with st.markdown('**Calculation time**'):
        # Tall
        predictions_test["Time Taken"] = [0 if i < 0 else i for i in predictions_test["Time Taken"] ]
        plt.figure(figsize=(3, 9))
        sns.set_theme(style="darkgrid")
        ax3 = sns.barplot(y=predictions_test.index, x="Time Taken", data=predictions_test)
    st.markdown(imagedownload(plt,'plot-calculation-time-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(9, 3))
    sns.set_theme(style="darkgrid")
    ax3 = sns.barplot(x=predictions_test.index, y="Time Taken", data=predictions_test)
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-calculation-time-wide.pdf'), unsafe_allow_html=True)


def filedownload(df, filename):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()  # strings <-> bytes conversions
    href = f'<a href="data:file/csv;base64,{b64}" download={filename}>Download {filename} File</a>'
    return href

def imagedownload(plt, filename):
    s = io.BytesIO()
    plt.savefig(s, format='pdf', bbox_inches='tight')
    plt.close()
    b64 = base64.b64encode(s.getvalue()).decode()  # strings <-> bytes conversions
    href = f'<a href="{b64}" download={filename}>Download {filename} File</a>'
    return href



with st.sidebar.header('File Uploader Section'):
    uploaded_file = st.sidebar.file_uploader("Upload an input as CSV file", type=["csv"])
```



```
    with st.sidebar.header('Set the optimization parameters\n (Grab the slider and set to any suitable point)'):

        split_size = st.sidebar.slider('Data split ratio (in fraction):', 0.0, 1.0, 0.7, 0.01)
        seed_number = st.sidebar.slider('Set the random-seed-value :', 0, 1, 100, 5)

    with st.sidebar.header('Project made by:'):
        st.write("Made by: MAINAK CHAUDHURI")

    #---------------------------------#

    st.subheader('Dataset display')



    if uploaded_file is not None:
        df = pd.read_csv(uploaded_file)
        st.markdown('**Snap of the dataset**')
        st.write(df)
        build_model(df)
    else:
        st.info('Upload a file')
        st.info('OR')
        if st.button('Use preloaded data instead'):
            st.info("Dataset used : Pima diabetes")


            diabetes = load_diabetes()

            X = pd.DataFrame(diabetes.data, columns=diabetes.feature_names).loc[:100]
            Y = pd.Series(diabetes.target, name='response').loc[:100]
            df = pd.concat( [X,Y], axis=1 )

            st.markdown('Displaying results form a sample preloaded data :')
            st.write(df.head(5))

            build_model(df)

elif option == 'Classification':
    st.subheader("Classifier ML Model Builder")

    def build_model(df):
        l = len(df)

        #df = df.iloc[:100]
        X = df.iloc[:,:-1] # Using all column except for the last column as X
        Y = df.iloc[:,-1] # Selecting the last column as Y


        st.markdown('**1.2. Dataset dimension**')
        st.write('X (Independent Axis)')
        st.info(X.shape)
        st.write('Y (Dependent Axis)')
        st.info(Y.shape)

        st.markdown('**1.3. Variable details**:')
        st.write('X variable (first few are shown)')
        st.info(list(X.columns[:int(l/5)]))
        st.write('Y variable')
        st.info(Y.name)

        # Build lazy model
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y,test_size = split_size,random_state = seed_number)
        clf = LazyClassifier(verbose=0,ignore_warnings=False, custom_metric=None)
```



```
    models_train,predictions_train = clf.fit(X_train, X_train, Y_train, Y_train)
    models_test,predictions_test = clf.fit(X_train, X_test, Y_train, Y_test)

    st.subheader('2.Model Performance Plot (Training Set)')

    st.write('Training set')
    st.write(predictions_train)
    st.markdown(filedownload(predictions_train,'training.csv'), unsafe_allow_html=True)

    st.write('Test set')
    st.write(predictions_test)
    st.markdown(filedownload(predictions_test,'test.csv'), unsafe_allow_html=True)

    st.subheader('3.Model Performance Plot(Test set)')


    with st.markdown('**Accuracy**'):
        # Tall
        predictions_test["Accuracy"] = [0 if i < 0 else i for i in predictions_test["Accuracy"] ]
        plt.figure(figsize=(5, 12))
        sns.set_theme(style="darkgrid")
        ax1 = sns.barplot(y=predictions_test.index, x="Accuracy", data=predictions_test)
        ax1.set(xlim=(0, 1))
    st.markdown(imagedownload(plt,'plot-r2-tall.pdf'), unsafe_allow_html=True)
        # Wide
    plt.figure(figsize=(12, 5))
    sns.set_theme(style="darkgrid")
    ax1 = sns.barplot(x=predictions_test.index, y="Accuracy", data=predictions_test)
    ax1.set(ylim=(0, 1))
    plt.xticks(rotation=90)
    st.pyplot(plt)
    st.markdown(imagedownload(plt,'plot-r2-wide.pdf'), unsafe_allow_html=True)



def filedownload(df, filename):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()  # strings <-> bytes conversions
    href = f'<a href="data:file/csv;base64,{b64}" download={filename}>Download {filename} File</a>'
    return href

def imagedownload(plt, filename):
    s = io.BytesIO()
    plt.savefig(s, format='pdf', bbox_inches='tight')
    plt.close()
    b64 = base64.b64encode(s.getvalue()).decode()  # strings <-> bytes conversions
    href = f'<a href="{b64}" download={filename}>Download {filename} File</a>'
    return href



with st.sidebar.header('File Uploader Section'):
    uploaded_file = st.sidebar.file_uploader("Upload an input as CSV file", type=["csv"])



with st.sidebar.header('Set the optimization parameters\n (Grab the slider and set to any suitable point)'):

    split_size = st.sidebar.slider('Data split ratio (in fraction):', 0.0, 1.0, 0.7, 0.01)
    seed_number = st.sidebar.slider('Set the random-seed-value :', 0, 1, 100, 5)

with st.sidebar.header('Project made by:'):
    st.write("Made by: MAINAK CHAUDHURI")

#---------------------------------#
```



```
    st.subheader('Dataset display')



    if uploaded_file is not None:
       df = pd.read_csv(uploaded_file)
       st.markdown('**Snap of the dataset**')
       st.write(df)
       build_model(df)
    else:
       st.info('Upload a file')
       st.info('OR')
       if st.button('Use preloaded data instead'):
          st.info("Dataset used : Pima diabetes")


          diabetes = load_diabetes()

          X = pd.DataFrame(diabetes.data, columns=diabetes.feature_names).loc[:100]
          Y = pd.Series(diabetes.target, name='response').loc[:100]
          df = pd.concat( [X,Y], axis=1 )

          st.markdown('Displaying results form a sample preloaded data :')
          st.write(df.head(5))

          build_model(df)

  elif option == 'Documentation':
     st.subheader("How to use Notebooker Pro")
     st.markdown('''The notebooker pro is a user-friendly software designed to help you make a good data science notebook in few
steps.
Well, notebooker pro will not be making a notebook for you, but will provide you with all the data insights that you
will need to put in your kernel. The notebooker pro has been provided with 4 major sections:

i.  **EDA (Explanatory Data Analysis)**  --> used to find important data and statistical insights from the uploaded files

ii. **Visualization** --> Used to perform data visualization with 5 basic important types of graphs

iii.**Regression** --> Loops through **30** different regression models and returns the complexity statistics of the result
                       of regression modelling for your dataset for chosen seed values and size. The only thing to keep in
                       mind while using this is that, the data must be fitting with a regression modelling. Datasets used
                       for classification algorithm might generate vague results. So use a proper dataset.
                       **[eg.: do not use iris,cancer,penguins etc. classifier dataset]**

iv. **Classification** --> Loops through **30** different classification models and returns the complexity statistics of the result
                       of classification modelling for your dataset for chosen seed values and size. The only thing to keep in
                       mind while using this is that, the data must be fitting with a classification modelling. Datasets used
                       for non-classification algorithm might generate vague results. So use a proper dataset.


**Features:**

**Upload file** => Upload only csv files.

**Data split**  => This is a linear slidebar, that will let you choose split ratio between 0 to 1

**Random seed** => Helps to randomize the data in training and testing data samples.
                   You may change to get the best accuracy of for a particular model.
                   ''')

elif option == 'About Us':
     st.subheader("About Us
")
     st.markdown('''This web application is made by Mainak Chaudhuri. He is a Computer Science and Engineering student of the
SRM University, studying in the second year of B.Tech. The main idea of this application is to help beginners and data science
enthusiasts chalk out a plan for preparing a good data science notebook, for college projects, online courses or to add in their
```

portfolio. This application accepts a dataset from the user and displays useful insights about the data. Additionally, it also helps the user visualize the data, choose the best supervised machine learning model (regression & classifaction handled separately) and decide the best suit depending on the dataset size,split and seed values which can be set by the user with the help of the side panel. This application claims to be the first of it's kind ever developed till date by a single developer and also has a serving history and positive reports from 180+ users.

N.B. : This application is an intellectual property of Mainak Chaudhuri and hence holds a reserved copyright. Any form of illegal immitation of graphics, contents or documentation without prior permission of the owner if proved, can result in legal actions against the plagiarist.'")

```
        st.success('For more info, feel free to contact @ : ')
        url = 'https://www.linkedin.com/in/mainak-chaudhuri-127898176/'

        if st.button('Mainak Chaudhuri'):
            webbrowser.open_new_tab(url)

if __name__ == '__main__':
    main()
```

6. CONCLUSION

This paper provides a detailed case study of the existing problems in data science and Notebooker Pro as a standing solution to all these problems, in a single software. The software, so made does not require any form of extensive training or technical know-how apart from a basic understanding statistics needed as a preliminary skill for data science.

7. FUTURE WORKS

This research paper and detailed case study can be a basis for future development of similar software with high-end production level deployment so that not only educators, but also industries, handling huge scale projects with plenty of data can be helped with such tools. Integration of more types of Machine Learning models like Reinforcement Learning models and also Deep Learning models dealing with NLP, GAN etc. can also be possible. This can also pave a way for the future of smart data management within the least possible deadline.

8. REFERENCES

[1] https://share.streamlit.io/mainakrepositor/notebooker-pro/app.py
[2] https://en.wikipedia.org/wiki/Regression_analysis
[3] https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/
[4] https://www.coursera.org/projects/data-science-streamlit-python
[5] https://searchbusinessanalytics.techtarget.com/definition/datasampling#:~:text=Data%20sampling%20is%20a%20statistical,larger%20data%20set%20being%20examined.

