Project Review Process - Instructions:

15-minute Review of Your Current Project and Status:

Steel Strength Data 2018
https://www.kaggle.com/datasets/fuarresvij/steel-test-data/data

The idea for my project were drawn from the "Steel Strength Data 2018" dataset available on Kaggle. This dataset provides detailed insights into various metal testing processes, crucial for analyzing the composition and properties of metals and their alloys. Metal testing can be categorized into destructive and nondestructive methods. Destructive testing typically dissects and examines an alloy's components to precisely determine its makeup, potentially destroying the original sample. Conversely, nondestructive testing retains the sample's integrity while identifying the metal's properties through accessible chemical-property databases. This enables the efficient identification of metals, preserving their original, reusable condition. My project leverages this comprehensive data to explore and potentially enhance methodologies for metal testing.

Brief Overview:

Title and Description of your Project and "Product":

Title:
Steel Strength;
Predictive Metallurgy: Modeling Mechanical Properties of Steel Alloys Using Advanced Data Analysis
Alloy Analyzer: Predictive Modeling of Casting Properties

Description:
The project is an advanced predictive tool designed to forecast the mechanical properties of steel castings based on their chemical compositions. Utilizing state-of-the-art machine learning algorithms, the system inputs raw material characteristics—specifically, the alloying elements like carbon, manganese, silicon, and others—and calculates key performance metrics of the final product, including yield strength, tensile strength , and elongation.

Product;
The product is a user-friendly application that serves metallurgists, materials engineers, and production specialists in the steel industry. It leverages a sophisticated model—potentially employing algorithms such as the Group Method of Data Handling (GMDH)—to ensure high predictive accuracy even with complex, non-linear data. This tool offers a streamlined interface where users can input alloy compositions and receive instant predictions, facilitating informed decision-making in alloy design and quality control. By incorporating features like data scaling and model training, "Alloy Analyzer…" stands as an indispensable asset in the

metallurgical field, optimizing material selection and process parameters to achieve desired casting qualities.

Version 1

A build a linear regression model that predicts mechanical properties of castings from their chemical compositions.

Step-by-Step Guide:

Libraries to use

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
```

Data Loading: Load the data from the CSV file.

Data Preprocessing:

- Drop non-numeric columns if they are not needed for modeling.
- Check for missing values and handle them (e.g., with imputation or removal).

Feature Selection: Identify which columns represent the chemical components (features) and which represent the mechanical properties (outputs).

Data Scaling: Scale the features due to the difference in ranges.

Model Training:

- Split the data into training and test sets.
- Train a linear regression model on the training set.

Model Evaluation:

- Evaluate the model on the test set using appropriate metrics.

Visualization:

- Plot the actual vs. predicted values for evaluation.

Coefficient Analysis: Analyze the model coefficients to understand the influence of each chemical component.

Code and brief description here

Version 2

Using GMDH algoritm

Group Method of Data Handling (GMDH) is a type of neural network that uses a self-organizing approach to model complex systems. It's particularly effective for discovering patterns and relationships in datasets with many variables and can be used for both regression and classification tasks.

Step-by-Step Guide:

- If steel alloy data is complex and non-linear, GMDH can model such non-linearities better than a linear regression model.
- GMDH can help in selecting the most significant features by automatically eliminating less important ones during the network construction process.
- With its ability to handle complex patterns, GMDH might provide better predictive accuracy compared to a simple linear regression model if the relationships in data are non-linear.
- GMDH networks aim to avoid overfitting by using a validation set to stop the network growth when the error on this set begins to increase.

Library to use:
`gmdhpy` - Integrate GMDH into code

Preprocessing such as GMDH can benefit from polynomial functions.
It is necessary to experiment with different GMDH-specific parameters to find the best model, such as the number of layers, neurons, and types of activation functions.
GMDH can be computationally intensive, especially when there are a large number of functions and complex models.
Although GMDH models can capture complex relationships, they tend to be less interpretable than linear models.

Before moving to GMDH, it is recommended to evaluate the performance of your current linear regression model and consider whether the increased complexity of GMDH is justified by the potential increase in forecasting accuracy.

Error: AttributeError: 'DataFrame' object has no attribute 'as_matrix'

The error is occurring because the `gmdhpy` package is trying to use the as_matrix() method which no longer exists in the newer versions of pandas. The use of `as_matrix()` needs to be replaced with `to_numpy()`.

- Locate the gmdh.py file in your site-packages directory (the path should be similar to C:\ProgramData\Anaconda3\envs\steel_strength\Lib\site-packages\gmdhpy\gmdh.py).
- Open gmdh.py in a text editor with administrator privileges.
- Search for .as_matrix() and replace it with .to_numpy().
- Save the file and try running your script again.

[Code and brief description here]

The developed code trained separate models for each of your target variables, and each has achieved a remarkably low Mean Squared Error (MSE).

MSE is a common measure of the accuracy of a regression model. It represents the average squared difference between the observed actual outcomes and the outcomes predicted by the

model. Lower values of MSE indicate better model performance, as this signifies that the predicted values are close to the actual values.
These are extremely low values for MSE, suggesting that models are performing very well in predicting the targets based on your features. This implies that the models can accurately capture the relationship between the features and the target variables. While low MSE is generally good, extremely low values could sometimes be a sign of overfitting, especially if the model performs significantly better on the training data compared to new, unseen data. Output shows the training time for each layer of the model. It's useful to see that most layers are trained fairly quickly, in about 0.20 to 0.24 seconds, which is efficient.

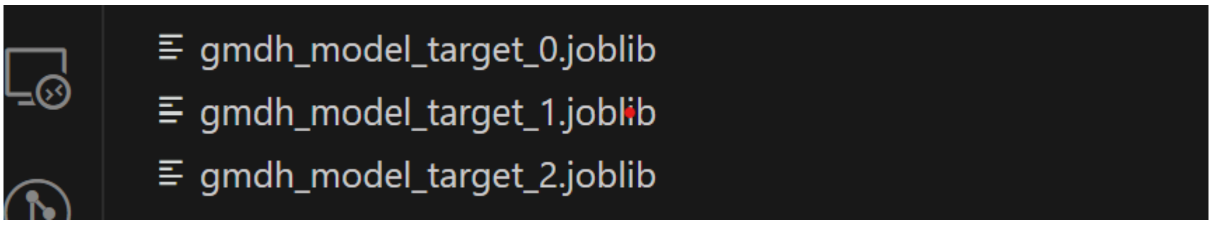
Step-by-step guide to serialize model using joblib, a library often used for this purpose because it efficiently handles large numpy arrays, which are common in machine learning models.

pip install joblib

Create a new Python file, such as `ModelSerialization.py`
Adapt your code with file `ModelSerialization.py`

```
#from ModelSerialization import save_model, load_model
```

After running `GMDH.py` your models will load in files.



```
gmdh_model_target_0.joblib
gmdh_model_target_1.joblib
gmdh_model_target_2.joblib
```

To load a model, use the `load` function from `joblib`. Create a file **load.py**

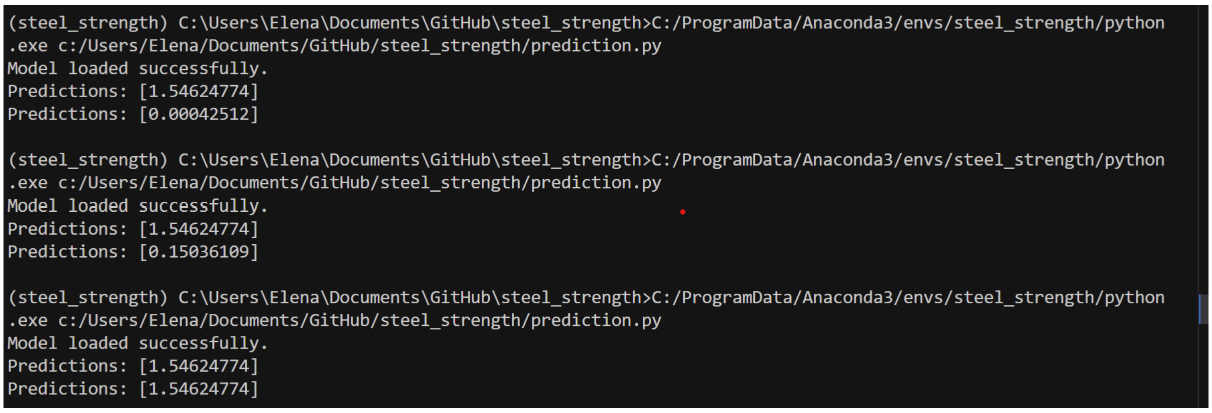
```
from joblib import load

# Load the model from the file
model = load('gmdh_model_target_2.joblib')
print("Model loaded successfully.")
```

A standalone Python script designed to load a previously trained and saved machine learning model from a .joblib file. After training and optimizing a model, saving it allows you to deploy the same model without needing to retrain it every time you start your application or process new data. It allows for quick testing and evaluation on new data.
Once the model is loaded, you can use it to make predictions on new data that has been processed in the same way as your training data.

Create file `prediction.py`

```
import pandas as pd
from joblib import load
```

```
# Load the model
model = load('gmdh_model_target_{0/1/2/..}.joblib')
print("Model loaded successfully.")
…..
…
…
```



```
(steel_strength) C:\Users\Elena\Documents\GitHub\steel_strength>C:/ProgramData/Anaconda3/envs/steel_strength/python
.exe c:/Users/Elena/Documents/GitHub/steel_strength/prediction.py
Model loaded successfully.
Predictions: [1.54624774]
Predictions: [0.00042512]

(steel_strength) C:\Users\Elena\Documents\GitHub\steel_strength>C:/ProgramData/Anaconda3/envs/steel_strength/python
.exe c:/Users/Elena/Documents/GitHub/steel_strength/prediction.py
Model loaded successfully.
Predictions: [1.54624774]
Predictions: [0.15036109]

(steel_strength) C:\Users\Elena\Documents\GitHub\steel_strength>C:/ProgramData/Anaconda3/envs/steel_strength/python
.exe c:/Users/Elena/Documents/GitHub/steel_strength/prediction.py
Model loaded successfully.
Predictions: [1.54624774]
Predictions: [1.54624774]
```

High-Level Requirements:

Version 0.NN - proof of concept(s)
Version 1.0 - "minimum viable product"
Version 1.NN.nn - near future release
Version x.NN.nn - future releases

Project Report: Machine Learning Model for Predicting Material Properties

Identifying the Real-world Objects:

- Objects (Nouns): Models, Scaler, Data, Predictions
- Functions (Verbs, Actions, Behavior): rain, Save, Load, Predict, Transform

Low-Level Requirements:

Input data must include features such as chemical components (in our case: C, Mn, Si, Cr, Ni, etc.) and target outputs (in our case: yield strength, tensile strength, elongation).
The model should accurately predict material properties based on input features.
Preprocessing (scaling) is required for input features.

Application Logic (workflow, dataflow, flowchart):

Package(s) Required: pandas
numpy
sklearn
joblib
gmdhpy

Data is loaded in .cvs file, features are selected, and data is split into training and testing sets.
Model Training: A GMDH-type neural network model is trained for each target output.
Model Saving: Each model is saved to a .joblib file using joblib.
The model is loaded from the .joblib file, new data is scaled using the saved scaler, and predictions are made.

Functions, Classes, Objects Required for Your Workflow:
Functions: train_model, save_model, load_model, predict
Classes: MultilayerGMDH
Objects: model, scaler

Environment and Tools Setup (instructions for someone else):
Python 3.8+
pip install pandas numpy scikit-learn joblib gmdhpy

Current Status of Coding and Testing:
Coding: Basic model training and saving functionalities are implemented. Prediction scripts are prepared.
Testing: Initial tests show that the models are saving and loading correctly. Predictive accuracy tests are ongoing.

Current Status of Documentation in Code:
Basic comments are included.
A README file is included.

Github Account Setup:
A GitHub repository is set up to manage version control and track changes.

Github Commits Done:
Initial commit with data preprocessing and model training scripts.
Added model saving and loading functionalities.
Updated scripts for better error handling and added documentation comments.

Remaining Tasks before "final product release"
    Extensions to V1.0