# Diamond sale prediction model synapsis

### About the data:

Has the following 24 features:

1. Cut
2. Color
3. Clarity
4. Carat\_weight
5. Cut\_quality
6. Lab
7. Symmetry
8. Polish
9. eye\_clean
10. culet\_size
11. culet\_condition
12. depth\_percent
13. table\_percent
14. meas\_length
15. meas\_width
16. meas\_depth
17. girdle\_min
18. girdle\_max
19. fluor\_color
20. fluor\_intensity
21. fancy\_color\_dominant\_color
22. fancy\_color\_secondary\_color
23. fancy\_color\_overtone
24. fancy\_color\_intensity

Has a lot of unknown values that changes the prediction and makes it less accurate.

## Models used

1. Linear Regression
2. ANN

### **Purpose of Using Two Models**

1. **Comparison of Model Performance**:
   * Linear Regression (a simple model) is often used as a baseline.
   * Neural Networks (complex models) are tested to see if they perform better, particularly when the relationship between features and the target variable is non-linear.

By using both models, you can analyze which one is more suitable for the dataset.

1. **Baseline vs. Advanced Modeling**:

* **Linear Regression**: Assumes a linear relationship between inputs and the target. It is interpretable and computationally inexpensive.
* **Neural Network**: Can capture more complex, non-linear relationships but requires more data, tuning, and computational power.

1. **Evaluate the Need for Complexity**:

* If the neural network doesn't significantly outperform linear regression, the simpler model might be sufficient.
* This helps in deciding whether the added complexity of deep learning is worth it for this specific dataset.

## Steps taken

 Data pre-processing will clean and prepare the data.

 Correlation heatmap will be displayed.

 The Linear Regression model will show intercept and coefficients.

 A Neural Network will train, and the loss history will be plotted.

 Predictions will be compared with actual values and visualized.

 The final trained model will be saved as tf\_m\_1.0.0.h5.

## R2 score

## Difficulties/ Problems faced

### For app.py

**Potential Issues & Fixes:**

1. **Label Encoding for Categorical Features:**
   * The code assumes that label encoders ({feature\_name}.joblib) for all categorical features exist in the specified path. If any encoder is missing, a FileNotFoundError will be raised.

**Fix:** Ensure all .joblib files are correctly generated and stored in C:/Projects/Diamond Sales Prediction/.

1. **Input Validation:**
   * There is no check for the length of the input list l. It must match the number of features.

**Fix:** Add a validation check for input length:

1. **String Handling in Categorical Features:**

* The handling of categorical features assumes all string inputs must be transformed using LabelEncoder.

**Suggestion:** Use try-except around the transformation in case an unseen category is passed, which can cause errors:

1. **Model Compilation:**

* Recompiling the model every time the script is loaded may not be necessary unless you plan to train the model further.

**Suggestion:** You can omit the model.compile() line unless training is required. For inference, the loaded model works without it.

1. **Path Handling:**

* Hardcoding the path (C:/Projects/Diamond Sales Prediction/) may lead to issues on different systems.

**Suggestion:** Use Path from pathlib to make the script portable

1. **Output Consistency:**

* The return value from model.predict is accessed via [0][0]. Ensure the model always returns a 2D array.

### For model.py

**Error: Missing or Incorrect Input Features**

* **Problem:** If the input list l does not match the features list in length or order, it will cause an error when reshaping or processing.
* **Fix:** Add a validation step at the beginning of the predict\_pipe function:

python

Copy code

**Error: Encoder File Missing**

* **Problem:** If any .joblib encoder file is missing, the program raises an unhandled exception.
* **Fix:** Provide a more informative error message and a fallback mechanism if desired:

python

Copy code

**Error: Incorrect Data Type Handling**

* **Problem:** The current implementation assumes all non-string inputs are numerical and converts them to float. If unexpected data types (e.g., None) are passed, it will fail.
* **Fix:** Add type validation and handle missing or invalid values

**Improvement: Numerical Feature Scaling**

* **Problem:** If the numerical features were scaled (e.g., Min-Max scaling) during model training, raw inputs might yield inaccurate predictions.
* **Fix:** Include the same preprocessing steps (e.g., scaling) used during training:

**Enhancement: Batch Prediction**

* **Problem:** The current implementation only supports single predictions. If you want to predict multiple inputs in one call, the function should handle batch input.
* **Fix:** Modify the input handling to accept multiple rows

**Error: Hardcoding File Paths**

* **Problem:** Hardcoding paths (e.g., "C:/Projects/Diamond Sales Prediction/") makes the code less portable.
* **Fix:** Use relative paths or environment variables

### Index.html

 **HTML Structure**

* Ensure the file is placed in the templates directory. Flask's render\_template function looks for files in the templates/ folder by default.

 **Flask Template Syntax**

* The template syntax {% if sale %} assumes that the sale variable is being passed correctly from the Flask app. If sale is None, the block will not render, which might make it seem like the page is blank.
* To debug, ensure sale is being set correctly in app.py.

 **Input Types**

* For numeric fields, use type="number" with proper validation to ensure users input valid data. You've done this for some fields, which is good.

 **Default Placeholder Values**

* Considered adding placeholder text or prefilled values that guide the user about the expected format.

### On the webpage

The error message "y contains previously unseen labels: 'None'" typically occurs when your input contains a value that the model or an encoder has not encountered during training. In this case, it seems that one of your categorical inputs (likely a string feature like cut, color, etc.) has been passed as 'None'.

**Possible Causes**

1. **Empty Form Fields**: If you left some form fields blank, Flask might be passing 'None' as a string or actual None to the prediction pipeline.
2. **Invalid Input Values**: Some categorical fields may have invalid values not seen during model training.

## Final changes made:

 **Error Handling**:

* Added error handling in both app.py and model.py to gracefully handle missing encoders or incorrect input formats.
* Displays error messages in the browser if prediction fails.

 **Form Inputs**:

* Refactored HTML to dynamically generate inputs using a loop in Jinja2 (index.html).

 **Prediction Pipeline**:

* Validated categorical and numeric inputs separately in model.py.
* Handled missing or incorrect encoder paths.

 **Debugging**:

* Flask app.py now passes errors to the template for easier debugging.
* Organized code for better maintainability and debugging.

## Additional features added

**Error Handling for Missing Form Data**

* If any form data is missing, the application will throw an error. Add proper handling for default or missing values.

**Debugging Predictions**

* Log the input and output to debug issues related to predictions.

**Flask Template Rendering**

* Ensure index.html exists in the templates directory and includes a placeholder for the prediction result ({{ sale }}).
* Example of a minimal index.html

 **Ensure All Required Files Are Present**

* Ensure all required encoders and the model file (tf\_m\_1.0.0.h5) are in the correct paths.

 **Start Flask Application Correctly**

* Run the application with the python app.py command in the project directory.
* Use app.run(debug=True) for development to capture and display errors in the browser.

 **Check the Browser Console**

* If the webpage is blank:
  + Open developer tools in your browser (F12).
  + Check the **Console** and **Network** tabs for errors.
  + Verify that the server is responding correctly to form submissions.

 **Handle Exceptions Globally**

* Add a Flask error handler to catch unhandled exceptions: