Week 4 — Data Acquisition & Description mporting all necessary

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler, PowerTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from scipy.sparse import hstack
from sklearn.compose import ColumnTransformer
import seaborn as sns
from sklearn.model_selection import GridSearchCV #for hypertuning
from sklearn.linear_model import LinearRegression,LogisticRegression, Lasso, Ridge
from lightgbm import LGBMRegressor
```

Importing csv File

```
# Uploading A csv file
import os, pandas as pd

CANDIDATES = ["exams.csv", "./exams.csv", "/content/exams.csv", "/mnt/data/exams.csv"]
df = None
for p in CANDIDATES:
    if os.path.exists(p):
        df = pd.read_csv(p); print(f"Loaded: {p}"); break

if df is None:
    try:
        from google.colab import files
        up = files.upload()
        fname = next(iter(up.keys()))
        df = pd.read_csv(fname)
        print(f"Uploaded and loaded: {fname}")
    except Exception:
        raise FileNotFoundError("Upload exams.csv or put it beside the notebook.")

df.head()
```

Choose Files student_per…ith_race.csv

student_performance_reduced_with_race.csv(text/csv) - 25236 bytes, last modified: 10/7/2025 - 100% done

```
Saving student_performance_reduced_with_race.csv to student_performance_reduced_with_race.csv
Uploaded and loaded: student_performance_reduced_with_race.csv
```

	math score	reading score	writing score	pass_flag	gender_encoded	lunch_standard	test preparation course_none	race/ethnicity_group B
0	72	72	74	1	1	1	1	1
1	69	90	88	1	1	1	0	0
2	90	95	93	1	1	1	1	1
3	47	57	44	0	0	0	1	0
4	76	78	75	1	0	1	1	0

Next steps: Generate code with df New interactive sheet

Double-click (or enter) to edit

```
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=df)
```

https://docs.google.com/spreadsheets/d/1Geg2P4IGDUbHL_5TpZufGcfkAmZq4AUGoJnhbVoCgkY/edit#gid=0

Using Gmail to run your business?
Get custom email, more storage, and 24/7 support with Google Workspace. Try Workspace

File Edit View Insert Format Data Tools Extensions Help

Menus 100% $ % .0 .00 123 Defaul... 10

A1 math score

	A	B	C	D	E	F	G
46	50	56	54	1	1	0	1
47	65	54	57	1	0	1	1
48	55	65	62	1	1	1	0
49	66	71	76	1	1	1	1
50	57	74	76	1	1	0	0
51	82	84	82	1	0	1	0
52	53	55	48	0	0	1	1
53	77	69	68	1	0	0	0
54	53	44	42	0	0	1	1
55	88	78	75	1	0	1	1
56	71	84	87	1	1	0	0
57	33	41	43	0	1	0	1
58	82	85	86	1	1	1	0
59	52	55	49	0	0	1	1
60	58	59	58	1	0	1	0
61	0	17	10	0	1	0	1
62	79	74	72	1	0	0	0

Sheet1

First let's analyze the dataset And Upload The csv File

```
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=df)
```

https://docs.google.com/spreadsheets/d/18hy8eAJF3xYZ6B4vUjqCz8uY_gzvgXCS3ZIUI0gZjaw/edit#gid=0

File Edit View Insert Format Data Tools Extensions Help

Menus 100% $ % .0 .00 123 Defaul... 10

C2 fx 74

	A	B	C	D	E	F	G
7	71	83	78	1	1	1	1
8	88	95	92	1	1	1	0
9	40	43	39	0	0	0	1
10	64	64	67	1	0	0	0
11	38	60	50	0	1	0	1
12	58	54	52	1	0	1	1
13	40	52	43	0	0	1	1
14	65	81	73	1	1	1	1
15	78	72	70	1	0	1	0
16	50	53	58	1	1	1	1
17	69	75	78	1	1	1	1
18	88	89	86	1	0	1	1
19	18	32	28	0	1	0	1
20	46	42	46	0	0	0	0
21	54	58	61	1	1	0	1
22	66	69	63	1	0	1	1
23	65	75	70	1	1	0	0

Sheet1

Here the dataset have categorical as well as numerical data. Categorical data includes race/ethinicity, parental level of education, lunch, test preparation course while numerical data includes math score, reading score, writing score.

```
df.shape, df.dtypes, df.isna().sum(), df.describe(include="all").T
```

```
((1000, 11),
 math score                     int64
 reading score                  int64
 writing score                  int64
 pass_flag                      int64
 gender_encoded                 int64
 lunch_standard                 int64
 test preparation course_none   int64
 race/ethnicity_group B         int64
 race/ethnicity_group C         int64
 race/ethnicity_group D         int64
 race/ethnicity_group E         int64
 dtype: object,
 math score                     0
 reading score                  0
 writing score                  0
 pass_flag                      0
 gender_encoded                 0
 lunch_standard                 0
 test preparation course_none   0
 race/ethnicity_group B         0
```

```
race/ethnicity_group C          0
race/ethnicity_group D          0
race/ethnicity_group E          0
dtype: int64,
                              count    mean        std   min    25%   50%  \
math score                   1000.0  66.089  15.163080   0.0  57.00  66.0
reading score                1000.0  69.169  14.600192  17.0  59.00  70.0
writing score                1000.0  68.054  15.195657  10.0  57.75  69.0
pass_flag                    1000.0   0.812   0.390908   0.0   1.00   1.0
gender_encoded               1000.0   0.518   0.499926   0.0   0.00   1.0
lunch_standard               1000.0   0.645   0.478753   0.0   0.00   1.0
test preparation course_none  1000.0   0.642   0.479652   0.0   0.00   1.0
race/ethnicity_group B       1000.0   0.190   0.392497   0.0   0.00   0.0
race/ethnicity_group C       1000.0   0.319   0.466322   0.0   0.00   0.0
race/ethnicity_group D       1000.0   0.262   0.439943   0.0   0.00   0.0
race/ethnicity_group E       1000.0   0.140   0.347161   0.0   0.00   0.0

                              75%    max
math score                   77.0  100.0
reading score                79.0  100.0
writing score                79.0  100.0
pass_flag                     1.0    1.0
gender_encoded                1.0    1.0
lunch_standard                1.0    1.0
test preparation course_none   1.0    1.0
race/ethnicity_group B        0.0    1.0
race/ethnicity_group C        1.0    1.0
race/ethnicity_group D        1.0    1.0
race/ethnicity_group E        0.0    1.0  )
```

Week 5 — Preprocessing & Feature Engineering

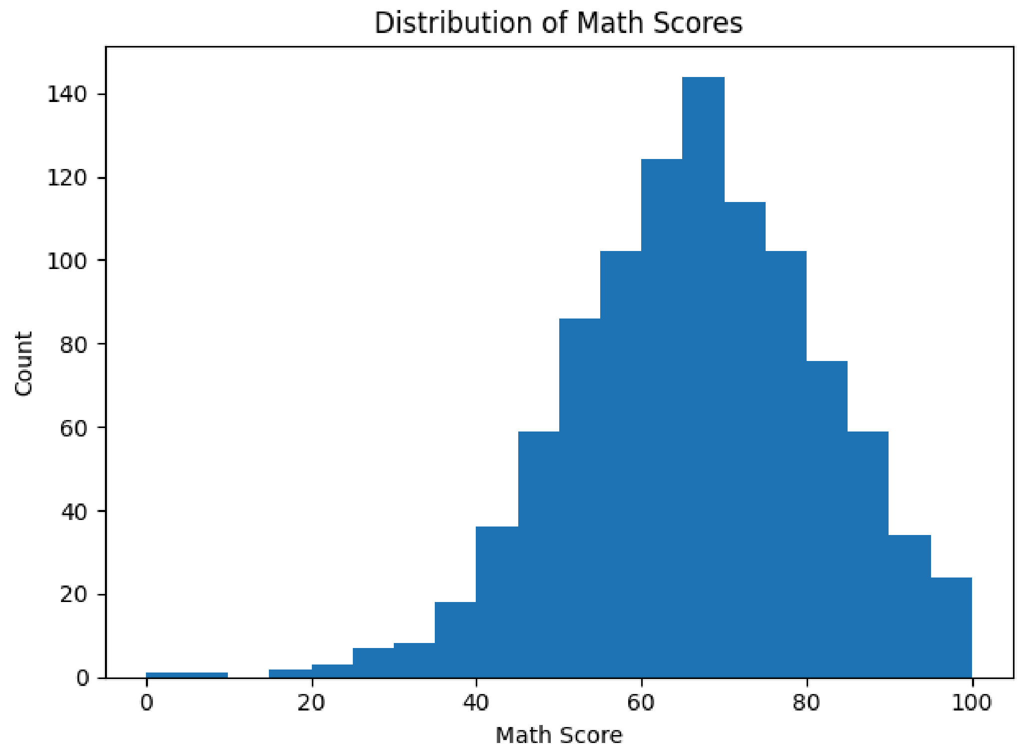
```
# Week 5 — Feature Engineering, Encoding, Split, Save
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# start from original data
work = df.copy()

# sanity check: required columns present
# The original categorical columns are already one-hot encoded, so this assertion is not needed.
# required = [
#     "gender","race/ethnicity","parental level of education",
#     "lunch","test preparation course",
#     "math score","reading score","writing score"
# ]
# missing = [c for c in required if c not in work.columns]
# assert not missing, f"Missing columns: {missing}"

# safe engineered features (no math used → no leakage)
work["avg_rw"] = work[["reading score","writing score"]].mean(axis=1)
work["gap_rw"] = work["reading score"] - work["writing score"]

# encode categoricals (simple + fine for this phase)
# The data is already one-hot encoded, so this step is not needed.
# cat_cols = ["gender","race/ethnicity","parental level of education","lunch","test preparation course"
# encoders = {}
# for c in cat_cols:
#     enc = LabelEncoder()
#     work[c] = enc.fit_transform(work[c].astype(str))
#     encoders[c] = enc
```

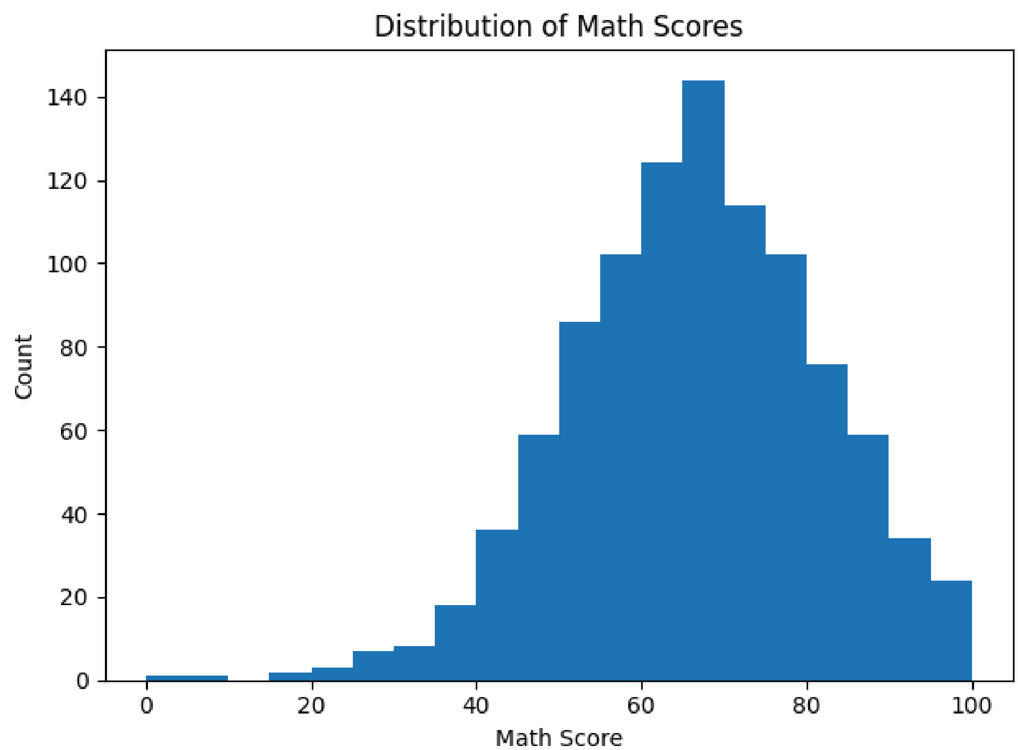
```
# 📎 70/15/15 split (Train/Val/Test)
X = work.drop(columns=["math score"])
y = work["math score"]

X_tmp, X_test, y_tmp, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_tmp, y_tmp, test_size=0.1765, random_state=42)  # ≈

# 💾 save preprocessed dataset (deliverable)
work.to_csv("student_performance_preprocessed_FIXED.csv", index=False)
print("Saved: student_performance_preprocessed_FIXED.csv")
```

```
Saved: student_performance_preprocessed_FIXED.csv
```

Week 6, Distributing Math Scores

```
import matplotlib.pyplot as plt

plt.figure()
plt.hist(work["math score"], bins=20)
plt.title("Distribution of Math Scores")
plt.xlabel("Math Score")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# optional: save for your report
plt.figure()
plt.hist(work["math score"], bins=20)
plt.title("Distribution of Math Scores")
plt.xlabel("Math Score")
plt.ylabel("Count")
plt.tight_layout()
plt.savefig("dist_math_score.png", dpi=200, bbox_inches="tight")
plt.show()
```





Displaying all the features from 0 to 999

Double-click (or enter) to edit

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column                          Non-Null Count  Dtype
```

```
 ---  ------                       --------------  -----
  0   math score                   1000 non-null   int64
  1   reading score                1000 non-null   int64
  2   writing score                1000 non-null   int64
  3   pass_flag                    1000 non-null   int64
  4   gender_encoded               1000 non-null   int64
  5   lunch_standard               1000 non-null   int64
  6   test preparation course_none  1000 non-null   int64
  7   race/ethnicity_group B       1000 non-null   int64
  8   race/ethnicity_group C       1000 non-null   int64
  9   race/ethnicity_group D       1000 non-null   int64
  10  race/ethnicity_group E       1000 non-null   int64
dtypes: int64(11)
memory usage: 86.1 KB
```

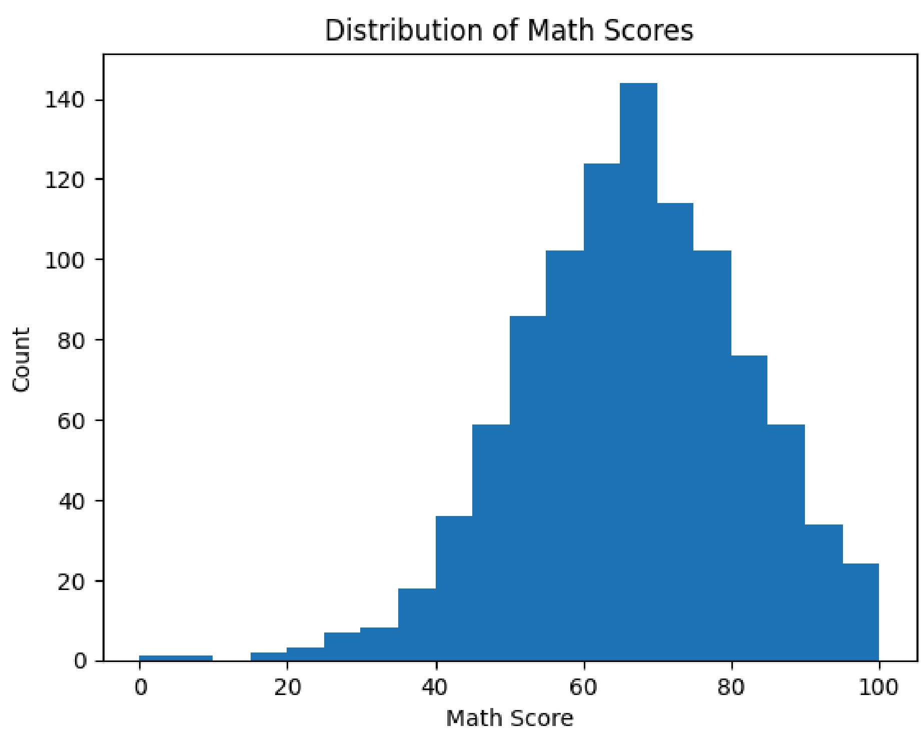
Week 6 — EDA & Baseline Model

Key Insights

- Students who engaged in test preparation** demonstrate **elevated math scores** (boxplot).
- **Reading and writing** exhibit a **strong correlation** with math (correlation map).
- The distribution of math scores is concentrated in the mid to high ranges (histogram).
- Baseline Linear Regression attains a robust R^2 with moderate MAE/RMSE.
- Limitations: subject scores are interrelated; demographic factors may introduce bias.

1. Distribution of Math *Scores*

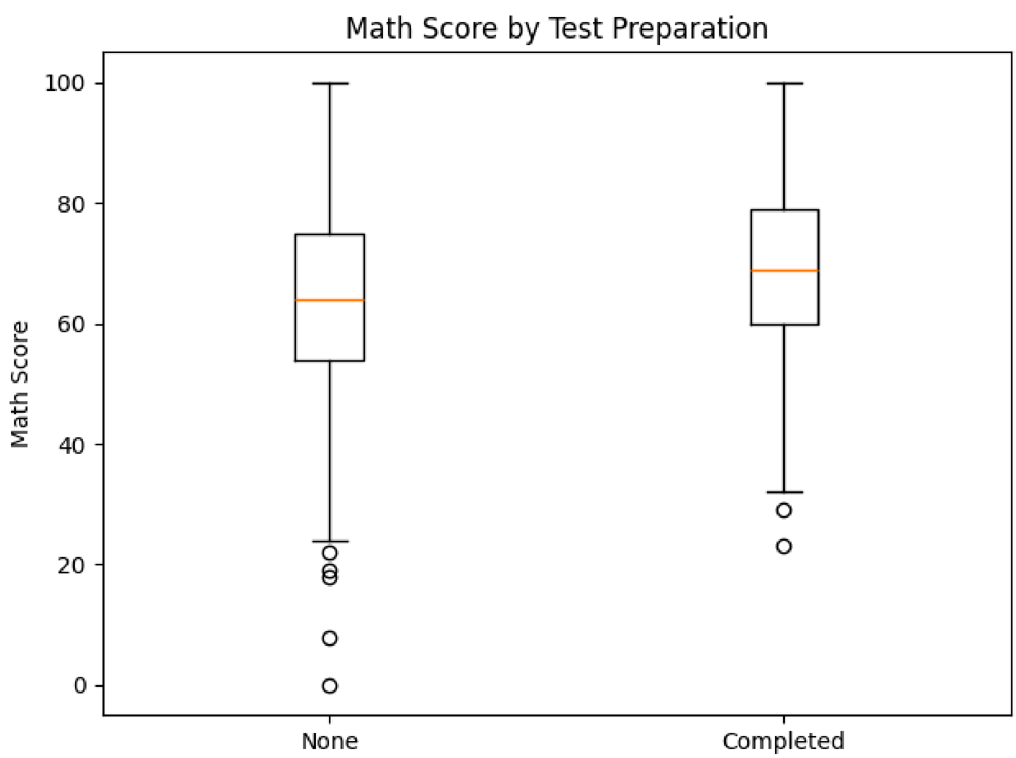
```
plt.figure()
plt.hist(work["math score"], bins=20)
plt.title("Distribution of Math Scores")
plt.xlabel("Math Score"); plt.ylabel("Count")
plt.show()
```



2. Math Score by Testing Preparation

```
import matplotlib.pyplot as plt

# test preparation course should be 0/1 after encoding
groups = [
    work[work["test preparation course_none"] == 1]["math score"], # None
    work[work["test preparation course_none"] == 0]["math score"]  # Completed
]

plt.figure()
plt.boxplot(groups, labels=["None", "Completed"])
plt.title("Math Score by Test Preparation")
plt.ylabel("Math Score")
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-1369810996.py:10: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() h
  plt.boxplot(groups, labels=["None", "Completed"])
```



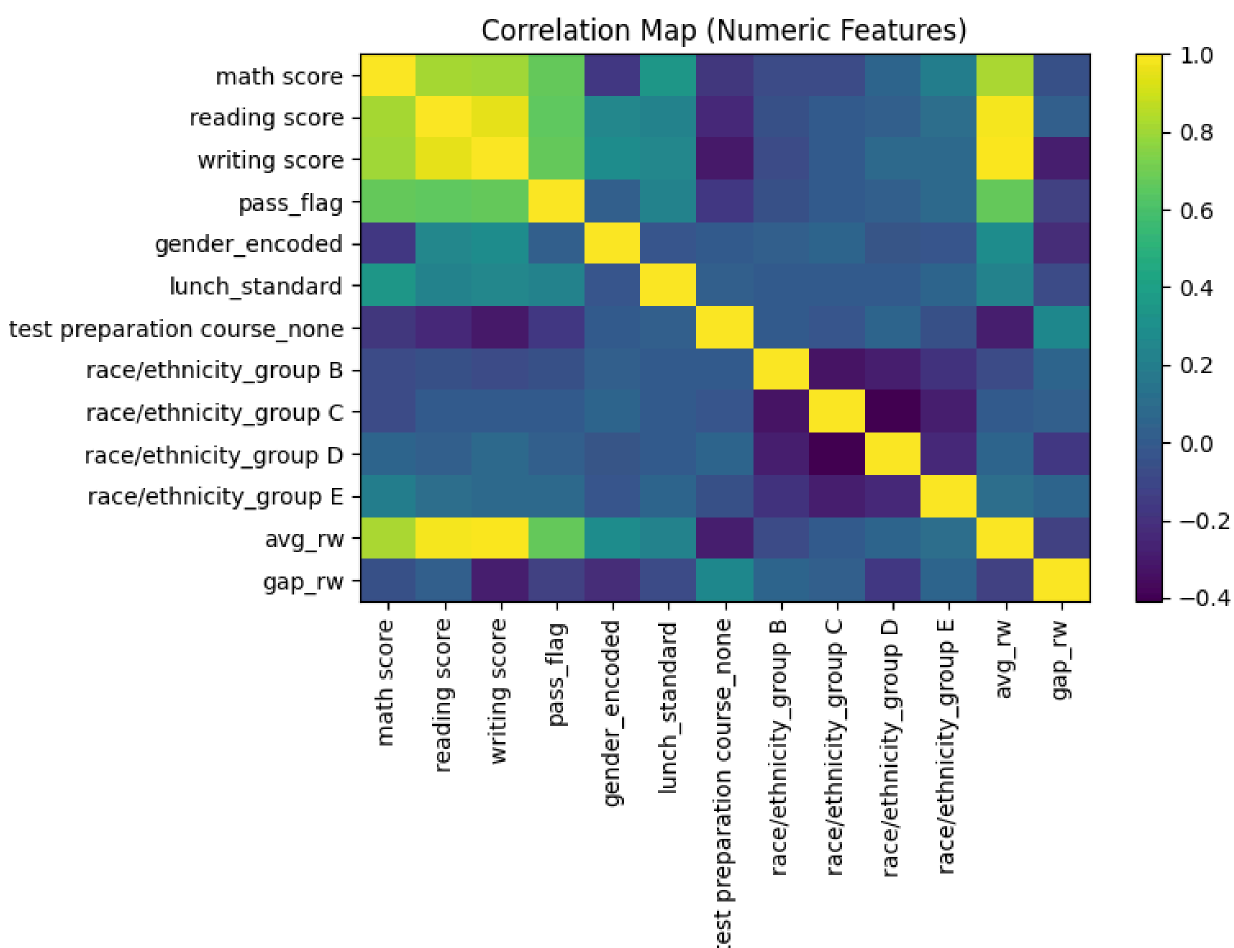

```
# --- Week 6: Baseline Linear Regression metrics (Validation & Test)

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np

lr = LinearRegression().fit(X_train, y_train)

def metrics(y_true, y_hat):
    r2 = r2_score(y_true, y_hat)
    mae = mean_absolute_error(y_true, y_hat)
```

```
    rmse = np.sqrt(((y_true - y_hat)**2).mean())
    return r2, mae, rmse

y_val_pred = lr.predict(X_val)
y_test_pred = lr.predict(X_test)

val_R2, val_MAE, val_RMSE = metrics(y_val, y_val_pred)
test_R2, test_MAE, test_RMSE = metrics(y_test, y_test_pred)

print('Validation → R²: %.3f | MAE: %.2f | RMSE: %.2f' % (val_R2, val_MAE, val_RMSE))
print('Test       → R²: %.3f | MAE: %.2f | RMSE: %.2f' % (test_R2, test_MAE, test_RMSE))
```

```
Validation → R²: 0.881 | MAE: 4.10 | RMSE: 5.03
Test       → R²: 0.898 | MAE: 3.96 | RMSE: 5.16
```

3. Correlational Map Of Numeric Features

```
corr_df = work.corr(numeric_only=True)
labels = corr_df.columns; corr = corr_df.values
plt.figure(figsize=(8,6))
im = plt.imshow(corr, aspect="auto")
plt.colorbar(im)
plt.xticks(range(len(labels)), labels, rotation=90)
plt.yticks(range(len(labels)), labels)
plt.title("Correlation Map (Numeric Features)")
plt.tight_layout()
plt.show()
```



4. Baseline of linear Regression and Metrics

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np

# fit baseline
lr = LinearRegression().fit(X_train, y_train)

# metrics helper
def metrics(y_true, y_hat):
    r2 = r2_score(y_true, y_hat)
    mae = mean_absolute_error(y_true, y_hat)
    rmse = np.sqrt(((y_true - y_hat)**2).mean())
    return r2, mae, rmse

# predict + evaluate
y_val_pred = lr.predict(X_val)
y_test_pred = lr.predict(X_test)

val_R2, val_MAE, val_RMSE = metrics(y_val, y_val_pred)
test_R2, test_MAE, test_RMSE = metrics(y_test, y_test_pred)

print("Validation → R²: %.3f | MAE: %.2f | RMSE: %.2f" % (val_R2, val_MAE, val_RMSE))
print("Test       → R²: %.3f | MAE: %.2f | RMSE: %.2f" % (test_R2, test_MAE, test_RMSE))
```

```
Validation → R²: 0.881 | MAE: 4.10 | RMSE: 5.03
Test       → R²: 0.898 | MAE: 3.96 | RMSE: 5.16
```

```
df.isna().any()
```

	0
math score	False
reading score	False
writing score	False
pass_flag	False
gender_encoded	False
lunch_standard	False
test preparation course_none	False
race/ethnicity_group B	False
race/ethnicity_group C	False
race/ethnicity_group D	False
race/ethnicity_group E	False

dtype: bool

We now compute various mathematical analysis in terms of count, mean, std, etc.

```
df.describe()
```

	math score	reading score	writing score	pass_flag	gender_encoded	lunch_standard	test preparation course_none	rac
count	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	
mean	66.08900	69.169000	68.054000	0.812000	0.518000	0.645000	0.642000	
std	15.16308	14.600192	15.195657	0.390908	0.499926	0.478753	0.479652	
min	0.00000	17.000000	10.000000	0.000000	0.000000	0.000000	0.000000	
25%	57.00000	59.000000	57.750000	1.000000	0.000000	0.000000	0.000000	
50%	66.00000	70.000000	69.000000	1.000000	1.000000	1.000000	1.000000	
75%	77.00000	79.000000	79.000000	1.000000	1.000000	1.000000	1.000000	
max	100.00000	100.000000	100.000000	1.000000	1.000000	1.000000	1.000000	

Key Insights

- Students who engaged in test preparation demonstrate elevated math scores (boxplot).
- Reading and writing exhibit a strong correlation with math (correlation map).
- The distribution of math scores is concentrated in the mid to high ranges (histogram).
- Baseline Linear Regression attains a robust R^2 with moderate MAE/RMSE.
- Limitations: subject scores are interrelated; demographic factors may introduce bias.

```
# This is formatted as code
```

Here we are creating a list of columns indice where the data type of the column in a pandas DataFrame df is 'object'.

```
# Identify the categorical features
cat_cols = [col for col in df.columns if df[col].dtype=='O']
cat_cols
```

```
[]
```

Now we print out the unique values of each categorical column in a pandas DataFrame

```
for col in cat_cols:
    print(df[col].unique())
```

To confirm that all the categorical columns in a pandas DataFrame can be converted to the categorical data type, you can check the number of unique values in each column and the percentage of unique values relative to the total number of values in the column. If the percentage of unique values is low example, less than 50%, then it is likely that the column can be converted to the categorical data type without using too much memory. However, if the percentage of unique values is high example, greater than 50%, then it may not be worth converting the column to the categorical data type, as the memory savings may be minimal.

```
# Get list of categorical columns
cat_cols = [col for col in df.columns if df[col].dtype == 'O']
```

```
# Loop over categorical columns
for col in cat_cols:
    unique_vals = df[col].nunique()
    total_vals = len(df[col])
    unique_pct = unique_vals / total_vals * 100
    print(f"{col}: {unique_vals} unique values ({unique_pct:.2f} of total)")
```

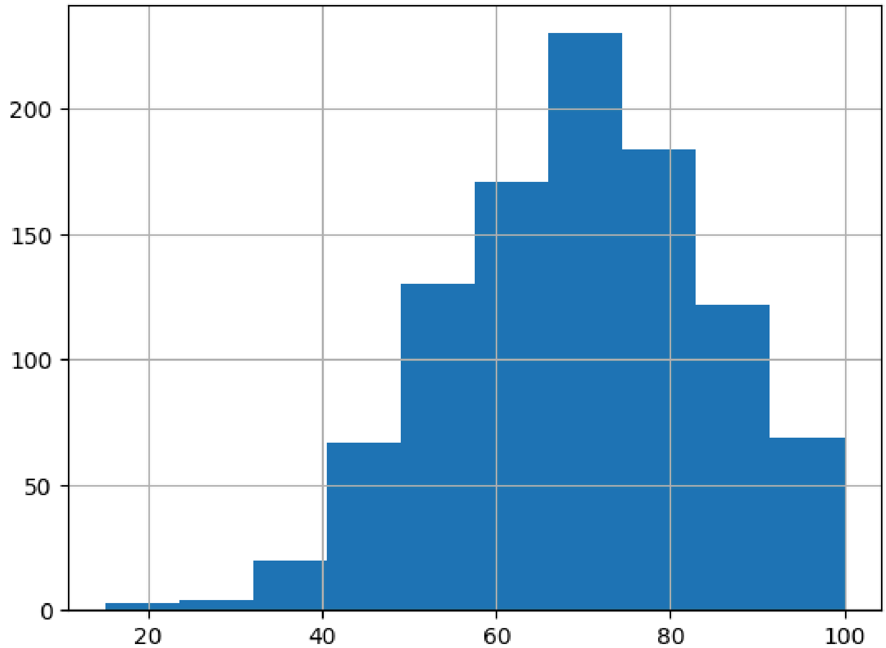
Now we convert all categorical columns in a Pandas DataFrame to the category data type, which can help reduce memory usage and potentially improve performance.

```
for col in cat_cols:
    df[col] = df[col].astype('category')
df.memory_usage(deep=True)
```

	0
Index	132
math score	8000
reading score	8000
writing score	8000
pass_flag	8000
gender_encoded	8000
lunch_standard	8000
test preparation course_none	8000
race/ethnicity_group B	8000
race/ethnicity_group C	8000
race/ethnicity_group D	8000
race/ethnicity_group E	8000

dtype: int64

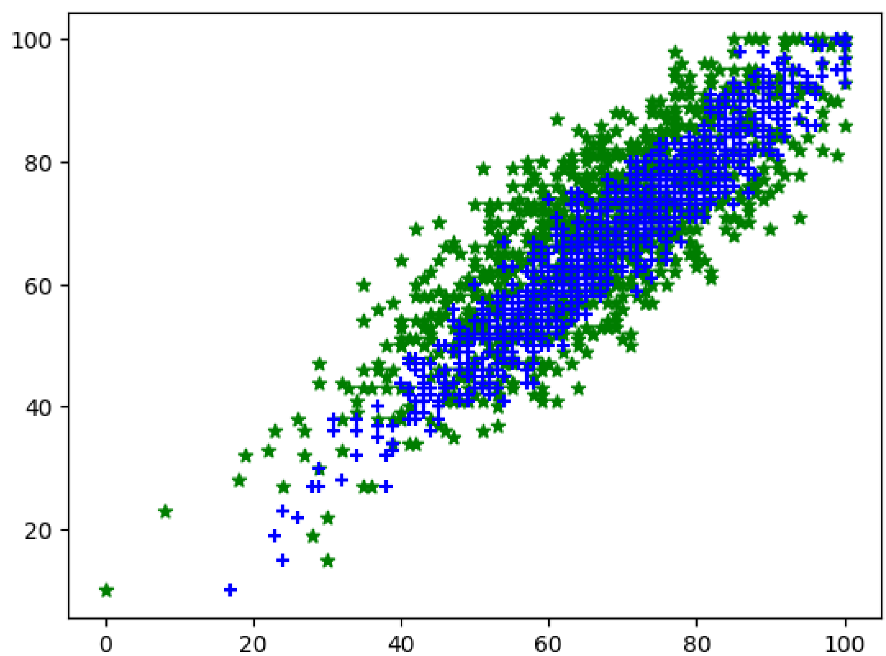
```
%matplotlib inline
# Creating Bar chart as the Target variable is Continuous
df['writing score'].hist();
```



Correlational Path Collection and Caculation

```
plt.scatter(df['math score'],df['writing score'],marker = '*', color = 'g')
plt.scatter(df['reading score'],df['writing score'],marker = '+', color = 'b')
```

```
<matplotlib.collections.PathCollection at 0x7fb13fbc9460>
```



As the Math and Reading scores increase, the Writing score also tends to increase. For this we calculate correaltion. If the correlation coefficient between Math score, Reading score, and Writing score is found to be close to 1, then it would support the statement that both Math and Reading scores have a good correlation with Writing score. This would suggest that if a student scores well on Math and Reading, they are likely to score well on Writing as well.

```
CorrelationData=df[['math score','reading score','writing score']].corr()
CorrelationData
```

	math score	reading score	writing score
math score	1.000000	0.817580	0.802642
reading score	0.817580	1.000000	0.954598
writing score	0.802642	0.954598	1.000000

Next steps: Generate code with CorrelationData New interactive sheet

```
# Based on the one-hot encoded columns present in the dataframe
# after the execution of cell ac8a4220 and cell 8013fab0.
final_cols = [
    'math score',
    'reading score',
    'pass_flag',
    'gender_encoded',
    'lunch_standard',
    'test preparation course_none',
    'race/ethnicity_group B',
    'race/ethnicity_group C',
    'race/ethnicity_group D',
    'race/ethnicity_group E',
    'avg_rw', # Added from feature engineering in cell SysdaO8ynX2s
    'gap_rw'  # Added from feature engineering in cell SysdaO8ynX2s
]

# Ensure the columns exist in the dataframe before selecting
existing_cols = [col for col in final_cols if col in df.columns]

df_final = df[existing_cols]
X = df_final.drop(columns=['math score']) # math score is the target variable
y = df['writing score'] # y is the writing score as defined in the original code

# Display X and y to verify
display(X.head())
display(y.head())
```

ender_encoded	lunch_standard	test preparation course_none	race/ethnicity_group B	race/ethnicity_
1	1	1	1	
1	1	0	0	
1	1	1	1	
0	0	1	0	
0	1	1	0	

```
num_cols = ['math score', 'reading score']
```

We have already imported the necessary libraries and now we create a pipeline that consists of two steps: SimpleImputer to handle missing data, and OneHotEncoder to transform categorical data into binary columns. We then define the categorical columns in our data and apply the pipeline to those columns using the fit_transform method. Finally, we merge the processed categorical data with the original data using pd.concat. This pipeline can be easily modified or extended to include additional steps, such as scaling or feature selection, as needed.

```
# Start clean from the raw csv or your earlier raw df
# df = pd.read_csv("exams.csv")    # uncomment if you need to reload raw data
work = df.copy()                    # make a working copy

categorical_cols = ['gender',
                    'race/ethnicity',
                    'parental level of education',
                    'lunch',
                    'test preparation course']

# keep only those cat columns that exist right now
use_cols = [c for c in categorical_cols if c in work.columns]
missing = [c for c in categorical_cols if c not in work.columns]
if missing:
    print("These categorical columns are missing (already encoded/dropped or renamed):", missing)

# now run your pipeline ONLY on present columns
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
```

```
])

if use_cols:
    cat_arr = categorical_pipeline.fit_transform(work[use_cols])
    cat_df = pd.DataFrame(cat_arr.toarray() if hasattr(cat_arr, "toarray") else cat_arr)
    work = pd.concat([work.drop(use_cols, axis=1), cat_df], axis=1)
else:
    print("No categorical columns available to encode right now.")
```

```
These categorical columns are missing (already encoded/dropped or renamed): ['gender', 'race/ethnicity',
No categorical columns available to encode right now.
```

Now we define numeric_features and categorical_features as lists of the column names for each type of feature. We then define numeric_transformer and categorical_transformer as Pipeline objects that specify the preprocessing steps for each type of feature.

Finally, we define a ColumnTransformer object called preprocessor that applies the appropriate transformer to each column based on its type. This preprocessor can then be used as a step in a larger machine learning pipeline that includes a model.

```
# define the preprocessing pipelines for numerical and categorical features
num_cols = ['math score', 'reading score']
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

categorical_cols = ['gender',
 'race/ethnicity',
 'parental level of education',
 'lunch',
 'test preparation course']

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder())])
```

```
# convert all column names to strings
df.columns = df.columns.astype(str)
df
```

	math score	reading score	writing score	pass_flag	gender_encoded	lunch_standard	test preparation course_none	race/ethnicity_grou
0	72	72	74	1	1	1	1	
1	69	90	88	1	1	1	0	
2	90	95	93	1	1	1	1	
3	47	57	44	0	0	0	1	
4	76	78	75	1	0	1	1	
...	...	...	...	...	...	...	...	

Next steps: Generate code with df New interactive sheet

Transforming Numerical numbers Using Matpolit

```
num_pipeline = Pipeline([
    ('num_smoothening',PowerTransformer())
])

# define the column transformer to preprocess both numeric and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols),
        ('cat', categorical_transformer, categorical_cols)])
```

Training and Testing

```
from sklearn.model_selection import train_test_split
X_train, X_test , y_train, y_test = train_test_split(X,y, test_size=0.2, random_state = 42)

# check the shapes of the training and test data
print(f'X_train shape: {X_train.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'X_test shape: {X_test.shape}')
print(f'y_test shape: {y_test.shape}')
X_train
```

```
X_train shape: (800, 12)
y_train shape: (800,)
X_test shape: (200, 12)
y_test shape: (200,)
```

	reading score	writing score	pass_flag	gender_encoded	lunch_standard	test preparation course_none	race/ethnicity_group B	rac
29	70	75	1	1	1	1	0	
535	83	83	1	1	0	0	0	
695	89	86	1	1	0	1	0	
557	67	66	1	0	0	1	0	
836	64	57	1	0	1	1	0	
...	...	...	...	...	...	...	...	
106	100	100	1	1	1	1	0	