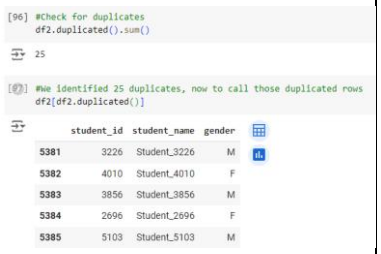
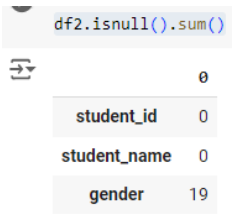
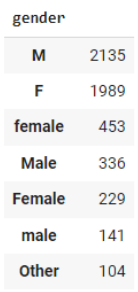
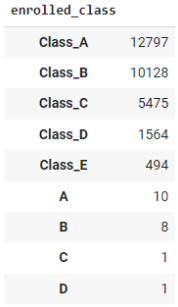
Task 2 Assignment

PART 1 – DATA QUALITY

1. **schools.csv – No data quality issues identified.**
2. **students.csv**

Data quality Issue	Screenshot of the issue	Code for data exploration	Code for data transformation	Justification
Duplicate rows (25)		df2.duplicated().sum() df2[df2.duplicated()]	df2.drop_duplicates(inplace=True) df2.duplicated().sum()	Removing duplicates is important for data accuracy; for e.g. keeping in duplicates can lead to inaccurate analysis and incorrect conclusions.
Null values within [‘gender’] variable		df2.isnull().sum()	df2['gender'].fillna('Other', inplace=True) df2.isnull().sum()	We replaced null values in [‘Gender’] with ‘Other’, since we did not have enough information to make inferences, and the count of null values was relatively low (19)
Value / formatting inconsistencies in [‘gender’]		df2['gender'].nunique() df2['gender'].value_counts()	Dict = {'Male':'M', 'Female':'F', 'male':'M', 'female':'F'} df2['gender'].replace(Dict, inplace=True) df2['gender'].value_counts()	We used dictionary to give consistent values, making 3 unique values in [‘Gender’]: ‘Male’, ‘Female’, and ‘Other’.

3. enrolments.csv

Data quality Issue	Screenshot of the issue	Code for data exploration	Code for data transformation	Justification
Formatting inconsistency – entries are made as regular integers as well as strings of characters to spell out the same integers	 enrolled_year_level 2 3274 1 3260 3 3229 4 3081 5 2994 6 2962 0 2848 7 2532 8 2115 9 1667 10 1234 11 838 12 429 six 3 one 3 eight 3 eleven 2 four 2 five 1 zero 1	df3['enrolled_year_level'].value_counts()	Dict = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6, 'seven':7, 'eight':8, 'nine':9, 'ten':10, 'eleven':11, 'twelve':12, 'zero':0} df3['enrolled_year_level'].replace(Dict, inplace=True) df3['enrolled_year_level'] = pd.to_numeric(df3['enrolled_year_level']) df3['enrolled_year_level'].value_counts().sort_index()	I created a dictionary of values for this column, matching all the alphabetic entries to their corresponding integers. I then replaced the alphabetic values with the integer dictionary. This formatting step will enable numerical operations and enhanced data analysis now that the column contains all int64 data type.
Formatting inconsistency – Equivalent values referred to variously as for e.g. 'Class_A' and 'A'	 enrolled_class Class_A 12797 Class_B 10128 Class_C 5475 Class_D 1564 Class_E 494 A 10 B 8 C 1 D 1	df3.nunique() df3['enrolled_class'].value_counts()	class_map = {'a': 'Class_A', 'b': 'Class_B', 'c': 'Class_C', 'd': 'Class_D', 'e': 'Class_E'} df3['enrolled_class'] = df3['enrolled_class'].map(class_map) df3['enrolled_class'].value_counts()	Used dictionary to replace entries for consistent formatting

4. year_level_mapping.csv – No data quality issues found

5. ae_assessment.csv

Data quality Issue	Screenshot of the issue	Code for data exploration	Code for data transformation	Justification
Formatting inconsistency in ['term_when_assessed'] – some entries are a string of alphanumeric, but could be an int64	term_when_assessed 1 121864 2 121860 4 100350 3 100340 Term 2 52 Term 1 48 Term 3 48 Term 4 38	df5['term_when_assessed'].value_counts()	term_dict = {'Term 1': 1, 'Term 2': 2, 'Term 3': 3, 'Term 4': 4} df5['term_when_assessed'].replace(term_dict, inplace=True) df5['term_when_assessed'] = pd.to_numeric(df5['term_when_assessed']) df5['term_when_assessed'].value_counts().sort_index()	Created a dictionary to match alphanumeric values to their equivalent integer values, then replace the former with the latter within the column. I also converted all entries to integer format with the pd.to_numeric() function, which will enable numerical operations and enhanced data analysis.
Missing values in ['result']	df5.isnull().sum() #'result' contains 1314 missing 0 id 0 assessment 0 assessment_id 0 student_id 0 school_id 0 year_level_when_assessed 0 year_when_assessed 0 term_when_assessed 0 result 1314	df5.isnull().sum()	df5['result'].fillna('not assessed', inplace=True) df5.isnull().sum()	Replaced all null values with 'not assessed'.

6. ae_assessment_mapping.csv – No data quality issues found

7. pat_assessment.csv

Data quality Issue	Screenshot of the issue	Code for data exploration	Code for data transformation	Justification

25 null value entries in ['assessment name'] and ['year']	`[ ] df7.isnull().sum()`
	`#We have 50 NaN values`
	
	0
	id 0
	assessment_code 0
	domain_id 0
	assessment_name 25
	student_id 0
	school_id 0
	year_level_when_assessed 0
	year 25
	scale_score 0
		df7.isnull().sum()	df7.dropna(inplace=True)
			df7.isnull().sum()
				The null values here made up a total of 50 of a total 52,720 entries in the data, a minimal amount; hence we are justified to drop all the null values without a significant impact on the overall data findings.

Entries in ['year'] are not in the correct format, they are dtype float64 but should be int64	`<class 'pandas.core.frame.DataFrame'>`
	`RangeIndex: 52720 entries, 0 to 52719`
	`Data columns (total 9 columns):`
	`# Column Non-Null Count Dtype`
	`0 id 52720 non-null int64`
	`1 assessment_code 52720 non-null object`
	`2 domain_id 52720 non-null int64`
	`3 assessment_name 52695 non-null object`
	`4 student_id 52720 non-null int64`
	`5 school_id 52720 non-null int64`
	`6 year_level_when_assessed 52720 non-null int64`
	`7 year 52695 non-null float64`
	`8 scale_score 52720 non-null object`
	`dtypes: float64(1), int64(5), object(3)`
	`memory usage: 3.6+ MB`
	
	year
	2015.0 5062
	2016.0 5929
	2017.0 6842
	2018.0 7841
	2019.0 8921
	2020.0 9019
	2021.0 9081
		df7['year'].value_counts().sort_index()	df7['year'] = df7['year'].astype('Int64')
			df7.info()
				I converted the columns to int64 because floating values are not necessary for a 'year' column, and this enables better accuracy and consistency across data variables.

['Scale score'] contains an invalid value 'not applicable'	4 33 3 33 2 30 1 11 Not Available 5 383 1 434 1	df7['scale_score'].value_counts()	df7.drop(df7[df7['scale_score'] == 'Not Available'].index, inplace=True) df7['scale_score'] = pd.to_numeric(df7['scale_score']) df7['scale_score'].value_counts().sort_index()	Remove invalid values to allow consistency within a single variable, this enables numerical operations and enhanced data analysis.
['Scale score'] contains outlier values	48 1241 49 1181 50 8403 316 1 320 1 324 1 383 1 434 1	df7['scale_score'].value_counts()	mean_score = round(df7.loc[df7['scale_score'] <= 50, 'scale_score'].mean()) df7.loc[df7['scale_score'] > 50, 'scale_score'] = mean_score df7['scale_score'].value_counts().sort_index()	Removing data outliers is important to maintain data quality because outliers can cause inaccuracies and distortions, and can lead to false conclusions from analysis.

8. naplan_assessment.csv

Data quality Issue	Screenshot of the issue	Code for data exploration	Code for data transformation	Justification

Duplicates	`#Check for duplicates`
	`df8.duplicated().sum()`
	`#Found 15 duplicates`
	`15`
		df8.duplicated().sum()
		
		df8[df8.duplicated()]
			df8.drop_duplicates(inplace=True)
			df8.duplicated().sum()
				Drop duplicates to increase data quality and accuracy, minimise any distortions

Format inconsistency in ['grade_level']	count grade_level 3 16143 5 14980 7 12660 9 8337 three 7 seven 3	df8['grade_level'].value_counts()	grade_dict = {'three':3, 'seven':7} df8['grade_level'].replace(grade_dict, inplace=True) df8['grade_level'] = pd.to_numeric(df8['grade_level']) df8['grade_level'].value_counts().sort_index()	Used dictionary to convert alphanumeric entries to integers; then converted whole column int64 format, for enhanced data manipulability.
['score'] contains out of bound values (greater than 700)	score 264 2561 7219 5128 11625 3171 14630 4273 23041 2550	df8.describe() bool = (df8['score'] > 700) OOB = df8['score'][bool] OOB	df8.drop(df8[df8['score'] > 700].index, inplace=True) df8['score'].value_counts().sort_index()	There were 5 outlier values in the dataset, and we dropped these entries as they were few enough not to impact data integrity.

9. naplan_mapping.csv – No data quality issues found

10. pat_mapping.csv – No data quality issues found

11. attendance.csv

Data quality Issue	Screenshot of the issue	Code for data exploration	Code for data transformation	Justification
Format inconsistency - ['year_level'] contains some entries expressed as alphabet instead of integer Value inconsistency - ['year_level'] may contain entries with trailing white spaces	4 428976 5 403336 5 459528 6 443104 6 402440 7 409112 7 435106 8 408136 8 347954 9 271100 9 308224 eight 8 eleven 1 five 6 four 4 nine 10 one 9	df11['year_level_when_attended'].value_counts().sort_index(key=lambda x: x.astype(str))	df11_dict = {'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'seven': 7, 'eight': 8, 'nine': 9, 'ten': 10, 'eleven': 11, 'twelve': 12} df11['year_level_when_attended'].replace(df11_dict, inplace=True) df11['year_level_when_attended'] = pd.to_numeric(df11['year_level_when_attended']) df11['year_level_when_attended'].value_counts().sort_index()	Re-formatted alphanumeric strings to integers with dictionary function; then converted the whole column to int64, to enhance data quality and consistency
Value inconsistency in [‘day of the week’]	day_of_week Wednesday 1848833 Thursday 1848831 Tuesday 1848830 Monday 1848827 Friday 1844915 Mon 13 Fri 11 Tue 10 Thu 9 Wed 7	df11['day_of_week'].value_counts()	day_dict = {'Monday': 'Mon', 'Tuesday': 'Tue', 'Wednesday': 'Wed', 'Thursday': 'Thu', 'Friday': 'Fri'} df11['day_of_week'].replace(day_dict, inplace=True) df11['day_of_week'].value_counts()	Re-format using dictionary function to give consistency across this variable.

PART 2 – JUSTIFYING THE DATASET

We have completed the data exploration and transformation step. The dataset is now fit for use for the specific needs of Edenglassie DoE.

We improved data quality by identifying issues and transforming the data to address these issues:

1. Duplicate entries – removed these to ensure data accuracy and prevent data distortions
2. Missing values – ensures data accuracy and quality. Missing values were addressed by dropping the entries entirely, or by imputing values e.g. mean from the column
3. Invalid values – created consistency within a single variable, this enables numerical operations and enhanced data analysis
4. Inconsistent formatting – fixed formatting to create consistency and enable numerical operations and enhanced data analysis. Int64 data type tends to be preferred for these purposes.
5. Outliers – removed these to maintain data quality; outliers can cause inaccuracies and distortions, which can lead to false conclusions from analysis.

The data is also fit for purpose because it meets the needs of our stakeholders.

- Data can now be used to track student performance, on an individual and group basis, and over a period of time, which will give insights into trends and patterns in performance.
- The data can also now be used to analyse gender balance and how it could be impacting student performance, which will create actionable insights for our client.

The dataset is therefore now fit for use to deliver the analytics solutions we will propose in the next step. We have ensured through the steps above that it is accurate, reliable, manipulable, comprehensive, and fit for purpose.

PART 3 – PIVOT TABLE OUTPUT

% Genders by Year Level			
	Female	Male	Other
8	53.2	44.7	2.1
9	41.7	58.3	0
10	49.0	44.9	6.1

Python code for my pivot table:

```
df_pivot_filtered = df_pivot[df_pivot['year_of_enrolment'] == 2020]
df_pivot_filtered = df_pivot_filtered[df_pivot_filtered['enrolled_school_id'] == 4]
df_pivot_filtered = df_pivot_filtered[df_pivot_filtered['enrolled_year_level'].isin([8, 9, 10])]

pivot_table = df_pivot_filtered.pivot_table(index='enrolled_year_level', columns='gender', aggfunc='size', fill_value=0)

pivot_table = pivot_table.div(pivot_table.sum(axis=1), axis=0) * 100

pivot_table = pivot_table.round(1)

pivot_table.columns.name = 'Gender %'

pivot_table
```