Student name: Nguyen Anh Tuan
Student ID: 103805949

Swinburne University of Technology

COS40007-Artificial Intelligence for Engineering

Portfolio 2

Nguyen Anh Tuan - 103805949

Studio class: Studio 1-1

Hanoi, Vietnam

Student name: Nguyen Anh Tuan
Student ID: 103805949

1. Dataset

The dataset for this portfolio is cement manufacturing in Studio 2 since I did Studio 2 in class. Furthermore, this dataset is also valuable for building models to predict and analyze the data on cement manufacturing elements. It also enables the investigation of correlations between elements, such as chemical composition and concrete strength, providing information for process optimization and sustainability in the building sector.

	A	B	C	D	E	F	G	H	I	J	K
1	cement	slag	ash	water	superplast	coarseagg	fineagg	age	strength		
2	141.3	212	0	203.5	0	971.8	748.5	28	29.89		
3	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51		
4	250	0	95.7	187.4	5.5	956.9	861.2	28	29.22		
5	266	114	0	228	0	932	670	28	45.85		
6	154.8	183.4	0	193.3	9.1	1047.4	696.7	28	18.29		
7	255	0	0	192	0	889.8	945	90	21.86		
8	166.8	250.2	0	203.5	0	975.6	692.6	7	15.75		
9	251.4	0	118.3	188.5	6.4	1028.4	757.7	56	36.64		
10	296	0	0	192	0	1085	765	28	21.65		
11	155	184	143	194	9	880	699	28	28.99		
12	151.8	178.1	138.7	167.5	18.3	944	694.6	28	36.35		
13	173	116	0	192	0	946.8	856.8	3	6.94		
14	385	0	0	186	0	966	763	14	27.92		
15	237.5	237.5	0	228	0	932	594	7	26.26		
16	167	187	195	185	7	898	636	28	23.89		
17	213.8	98.1	24.5	181.7	6.7	1066	785.5	100	49.97		
18	237.5	237.5	0	228	0	932	594	28	30.08		
19	336	0	0	182	3	986	817	28	44.86		
20	190.7	0	125.4	162.1	7.8	1090	804	3	15.04		
21	312.7	0	0	178.1	8	999.7	822.2	28	25.1		
22	229.7	0	118.2	195.2	6.1	1028.1	757.6	3	13.36		
23	228	342.1	0	185.7	0	955.8	674.3	7	21.92		
24	236	157	0	192	0	972.6	749.1	7	20.42		
25	132	207	161	179	5	867	736	28	33.3		
26	331	0	0	192	0	1025	821	28	31.74		
27	310	143	0	168	10	914	804	28	45.3		

2. EDA

a. Variable Identification

- Target variable: 'strength'
- Predictors (Input variables): 'cement', 'slag', 'ash', 'water', 'superplastic', 'coarseagg', 'fineagg', 'age'

b. Distribution of Label Categories

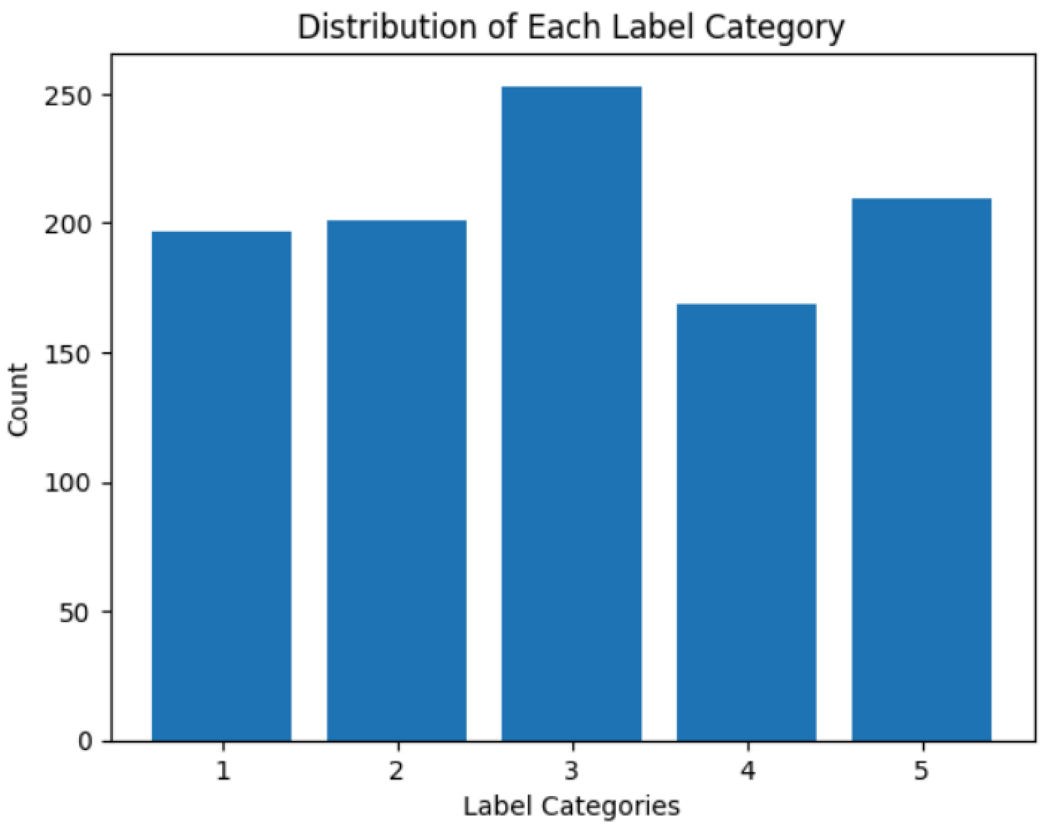
Here are the code and the visualization for the label categories.

Student name: Nguyen Anh Tuan
Student ID: 103805949

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.bar(cement['label'].value_counts().index, cement['label'].value_counts().values)

# Add labels and title
ax.set_xlabel('Label Categories')  # x-axis label
ax.set_ylabel('Count')         # y-axis label
ax.set_title('Distribution of Each Label Category')  # plot title

# Display the plot
plt.show()
```



The bar chart shows the "Distribution of Each Label Category" with five distinct label categories (1-5) on the x-axis and their corresponding counts on the y-axis. Category 3 shows the highest frequency with approximately 250 counts, while Category 4 has the lowest with about 170 counts. The other categories (1, 2, and 5) maintain relatively similar frequencies, hovering around 200 counts each.

c. Distribution of Age Categories

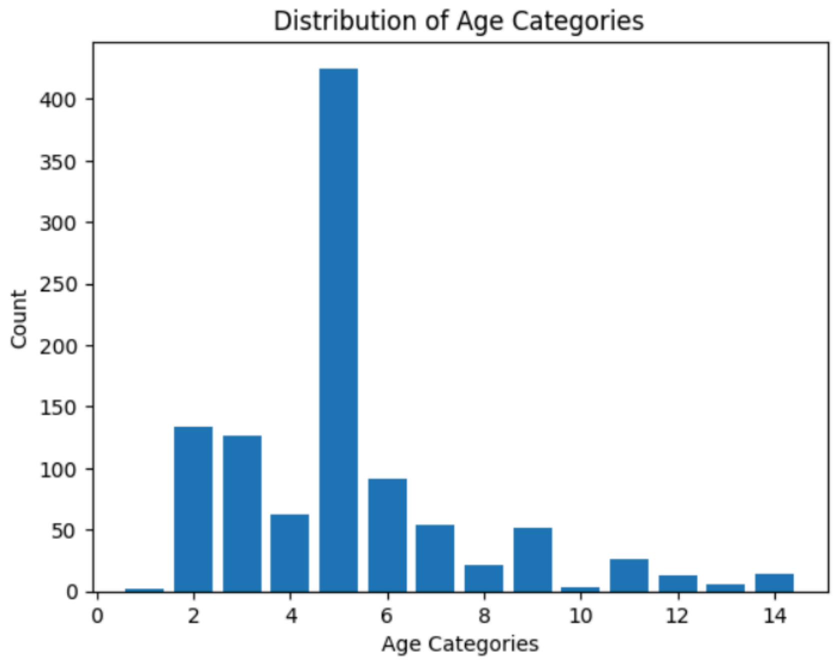
Here is the code and the visualization for this part.

Student name: Nguyen Anh Tuan
Student ID: 103805949

```
fig, ax = plt.subplots()
ax.bar(cement['age'].value_counts().index, cement['age'].value_counts().values)

ax.set_xlabel('Age Categories')  # x-axis label for age categories
ax.set_ylabel('Count')       # y-axis label for frequency of each category
ax.set_title('Distribution of Age Categories')  # plot title

# Display the plot
plt.show()
```



The bar chart illustrates the "Distribution of Age Categories" and presents a notably different pattern. It shows a right-skewed distribution across age categories from 0 to 14, with a pronounced peak at age category 5, reaching about 420 counts. There's a sharp decline in frequency after this peak, with the older age categories (8-14) showing significantly lower counts, generally below 50. The younger age categories (2-4) show moderate frequencies between 50 and 150 counts.

3. Class labeling for target variable / developing ground truth data

The target variable "strength" was categorized into five classes using a Python function to develop ground truth data. The function labels the performance as Very Low (1) for strength below 20, Low (2) for strength between 20 and 30, Moderate (3) for strength between 30 and 40, Strong (4) for strength between 40 and 50, and Very strong (5) for strength over 50. This class labeling ensures the data is structured for classification tasks and allows for clearer insights into concrete's strength.

Student name: Nguyen Anh Tuan
Student ID: 103805949

```
def label_strength(i):
    if i < 20:
        i =1
    elif 20 <= i < 30:
        i=2
    elif 30 <= i < 40:
        i=3
    elif 40 <= i < 50:
        i=4
    else:
        i=5
    return i
```

```
cement['label']=cement['strength'].apply(label_strength)

data_dict = cement.groupby('label')


cement.head()
```

[6]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength	label
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89	2
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51	2
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22	2
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85	4
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29	1

4. Feature engineering and Feature selection

a. Categorize Performance

A “label_strength()” function assigns labels (1-5) to concrete based on their "strength" creating a new column, label.

Student name: Nguyen Anh Tuan
Student ID: 103805949

```
def label_strength(i):
    if i < 20:
        i =1
    elif 20 <= i < 30:
        i=2
    elif 30 <= i < 40:
        i=3
    elif 40 <= i < 50:
        i=4
    else:
        i=5
    return i
```

```
cement['label']=cement['strength'].apply(label_strength)

data_dict = cement.groupby('label')


cement.head()
```

[6]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength	label
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89	2
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51	2
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22	2
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85	4
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29	1

b. Normalization

Min-Max scaling is used to normalize the chosen columns in the cement dataset. It converts each value in the designated columns ('cement','slag', 'ash', etc.) to a range between 0 and 1 by using the formula (x - x.min()) / (x.max() - x.min()). By guaranteeing that every feature has the same scale, this method makes them more appropriate for machine learning techniques that are sensitive to feature scaling.

```
#normalize using minmaxscale

df_min_max_scale = ['cement', 'slag', 'ash', 'water', 'superplastic', 'coarseagg','fineagg']
cement[df_min_max_scale] = cement[df_min_max_scale].apply(lambda x: (x - x.min()) / (x.max() - x
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

```
e using minmaxscale

x_scale = ['cement', 'slag', 'ash', 'water', 'superplastic', 'coarseagg','fineagg']
_min_max_scale] = cement[df_min_max_scale].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
```

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength	label
0	0.089726	0.589872	0.000000	0.652556	0.000000	0.496512	0.387607	5	29.89	2
1	0.152740	0.117418	0.621189	0.291534	0.335404	0.813372	0.507275	4	23.51	2
2	0.337900	0.000000	0.478261	0.523962	0.170807	0.453198	0.670346	5	29.22	2
3	0.374429	0.317195	0.000000	0.848243	0.000000	0.380814	0.190667	5	45.85	4
4	0.120548	0.510295	0.000000	0.571086	0.282609	0.716279	0.257652	5	18.29	1

c. Composite features

The covariance between pairs of columns in the cement dataset is used to produce composite features. In the case of the 'cement' and'slag' columns, for instance, the covariance is the new feature cement_slag. Similar calculations are done to determine the covariance between each pair of columns to construct other composite characteristics such as cement_ash, water_fineagg, and ash_superplastic. To represent the interactions between various material qualities in the dataset, these new features use covariance, which assesses how two variables change together.

```
# Composite features
cement['cement_slag'] = cement['cement'].cov(cement['slag'])
cement['cement_ash'] = cement['cement'].cov(cement['ash'])
cement['water_fineagg'] = cement['water'].cov(cement['fineagg'])
cement['ash_superplastic'] = cement['ash'].cov(cement['superplastic'])
cement.head()
```

[15]:

ı	water	superplastic	coarseagg	fineagg	age	strength	label	cement_slag	cement_ash	water_fineagg	ash_superplastic
)	0.652556	0.000000	0.496512	0.387607	5	29.89	2	-0.015764	-0.030331	-0.015461	0.022399
)	0.291534	0.335404	0.813372	0.507275	4	23.51	2	-0.015764	-0.030331	-0.015461	0.022399
1	0.523962	0.170807	0.453198	0.670346	5	29.22	2	-0.015764	-0.030331	-0.015461	0.022399
)	0.848243	0.000000	0.380814	0.190667	5	45.85	4	-0.015764	-0.030331	-0.015461	0.022399
)	0.571086	0.282609	0.716279	0.257652	5	18.29	1	-0.015764	-0.030331	-0.015461	0.022399

Student name: Nguyen Anh Tuan
Student ID: 103805949

d. Feature selection

Except for 'cement', 'water', 'superplastic', and 'age', other features have weak relationships with the 'strength' feature, and they don't consider statistical decision-making (correlation). Therefore, I dropped those weak features and maintained the other 4 features.

```
corr_dict = ['cement', 'water', 'superplastic', 'age']
corr = cement[corr_dict].corrwith(cement['strength'])
print(corr)
```

```
cement          0.497832
water          -0.289633
superplastic    0.366079
age             0.474690
dtype: float64
```

For the feature selection, here is the code about it.

- Selected features with normalization

```
corr_dict = ['cement', 'water', 'superplastic', 'age', 'cement_slag', 'cement_ash', 'water_fineag
select_feature_concrete = cement[corr_dict]
select_feature_concrete.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   cement            1030 non-null   float64
 1   water             1030 non-null   float64
 2   superplastic      1030 non-null   float64
 3   age               1030 non-null   int64
 4   cement_slag       1030 non-null   float64
 5   cement_ash        1030 non-null   float64
 6   water_fineagg     1030 non-null   float64
 7   ash_superplastic  1030 non-null   float64
 8   strength          1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

[98]:

```
select_feature_concrete.head()
```

[98]:

	cement	water	superplastic	age	cement_slag	cement_ash	water_fineagg	ash_superplastic	strength	label
0	0.089726	0.652556	0.000000	5	-0.015764	-0.030331	-0.015461	0.022399	29.89	2
1	0.152740	0.291534	0.335404	4	-0.015764	-0.030331	-0.015461	0.022399	23.51	2
2	0.337900	0.523962	0.170807	5	-0.015764	-0.030331	-0.015461	0.022399	29.22	2
3	0.374429	0.848243	0.000000	5	-0.015764	-0.030331	-0.015461	0.022399	45.85	4
4	0.120548	0.571086	0.282609	5	-0.015764	-0.030331	-0.015461	0.022399	18.29	1

Student name: Nguyen Anh Tuan
Student ID: 103805949

]:
```
select_feature_concrete.to_csv('selected_feature_concrete.csv')
```

- Selected feature without normalization

[100]:
```
cemented = pd.read_csv('convert_concrete.csv')

cemented.head()
```

[100…

	Unnamed: 0	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength	label
0	0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89	2
1	1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51	2
2	2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22	2
3	3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85	4
4	4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29	1

[101]:
```
cemented.to_csv('selected_converted_concrete.csv')
```

5. Training and decision tree model development

I also create new 4 datasets besides 'convert_concrete.csv' to train the decision tree model with a 70-30% split for training and validation data for all the cases

- normalized_concrete.csv: all features with normalization and without composite features.
- features_concrete.csv: all features with normalization and containing composite features.
- selected_feature_concrete.csv: selected features with normalization.
- selected_converted_concrete.csv: selected feature without normalization.

Here is the code for training the decision tree model

Student name: Nguyen Anh Tuan
Student ID: 103805949

```
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

concrete = pd.read_csv("/kaggle/working/features_concrete.csv")

feature_cols = ['cement', 'water', 'superplastic', 'age']
x = concrete[feature_cols] # Feature
y = concrete['label']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

clf = DecisionTreeClassifier()

clf = clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
```

```
# Train Model Decision Tree with all features with normalisation and without composite features
concrete1 = pd.read_csv("/kaggle/working/normalized_concrete.csv")

x1 = concrete1[feature_cols] # Feature
y1 = concrete1['label']
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.3, random_state = 1)

clf = DecisionTreeClassifier()

clf = clf.fit(x1_train, y1_train)

y1_pred = clf.predict(x1_test)

print("Accuracy: ", metrics.accuracy_score(y1_test, y1_pred))
```

```
# selected features with normalisation
concrete2 = pd.read_csv("/kaggle/working/features_concrete.csv")

x2 = concrete2[feature_cols] # Feature
y2 = concrete2['label']
x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y2, test_size=0.3, random_state = 1)

clf = DecisionTreeClassifier()

clf = clf.fit(x2_train, y2_train)

y2_pred = clf.predict(x2_test)

print("Accuracy: ", metrics.accuracy_score(y2_test, y2_pred))
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

```
# selected features with normalisation
concrete3 = pd.read_csv("/kaggle/working/selected_feature_concrete.csv")

x3 = concrete3[feature_cols] # Feature
y3 = concrete3['label']
x3_train, x3_test, y3_train, y3_test = train_test_split(x3, y3, test_size=0.3, random_state = 1)

clf = DecisionTreeClassifier()

clf = clf.fit(x3_train, y3_train)

y3_pred = clf.predict(x3_test)

print("Accuracy: ", metrics.accuracy_score(y3_test, y3_pred))
```

```
# selected feature without normalisation
concrete4 = pd.read_csv("/kaggle/working/selected_converted_concrete.csv")

x4 = concrete4[feature_cols] # Feature
y4 = concrete4['label']
x4_train, x4_test, y4_train, y4_test = train_test_split(x4, y4, test_size=0.3, random_state = 1)

clf = DecisionTreeClassifier()

clf = clf.fit(x4_train, y4_train)

y4_pred = clf.predict(x4_test)

print("Accuracy: ", metrics.accuracy_score(y4_test, y4_pred))
```

The results for these models are concluded in the table in the next section.

6. Final comparison table

Model 1 (‘convert_concrete.csv’)	**Model 2 (‘normalized_concrete.csv’)**	**Model 3 (‘features_concrete.csv’)**	**Model 4 (‘selected_feature_concrete.csv’)**	**Model 5 (‘selected_converted_concrete.csv’)**
62.13592233009708%	62.45954692556634%	63.43042071197411%	62.7831715210356%	63.43042071197411%

7. Summary

When the five models are compared, their performance levels vary, peaking at 63.43%. Both Model 3 and Model 5 achieve this maximum accuracy, indicating that in some cases, the feature selection and transformation processes could improve performance. Accuracy is marginally lower for Models 1, 2, and 4, with Model 1 attaining 62.14% and Model 4 62.78%. Overall, none of the models' performances have significantly improved or decreased, suggesting that the modifications made to the feature processing procedures do not result in appreciable variations in accuracy.

8. Appendix:

- Source code for the portfolio 2: https://www.kaggle.com/code/twananguyen/studio-2-port-2-submit
- Link for the dataset: https://www.kaggle.com/datasets/vinayakshanawad/cement-manufacturing-concrete-dataset