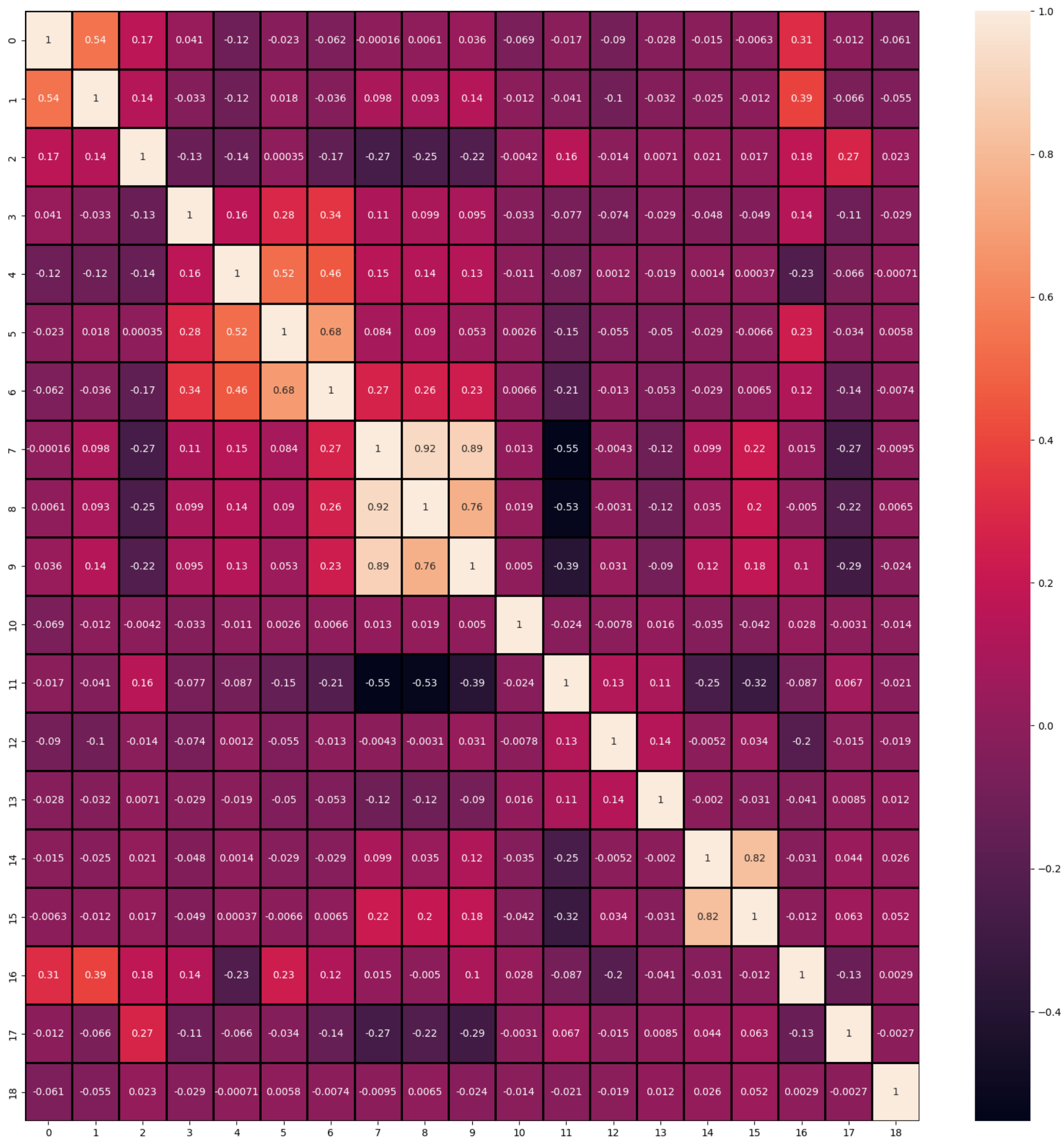
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('data_melbourne.csv')

data.head()
```

```
   Unnamed: 0  Average Outflow  Average Inflow  Energy Consumption
Ammonia  \
0           0            2.941           2.589              175856
27.0
1           1            2.936           2.961              181624
25.0
2           2            2.928           3.225              202016
42.0
3           3            2.928           3.354              207547
36.0
4           4            2.917           3.794              202824
46.0

   Biological Oxygen Demand  Chemical Oxygen Demand  Total Nitrogen  \
0                     365.0                   730.0          60.378
1                     370.0                   740.0          60.026
2                     418.0                   836.0          64.522
3                     430.0                   850.0          63.000
4                     508.0                  1016.0          65.590

   Average Temperature  Maximum temperature  Minimum temperature  \
0                 19.3                 25.1                 12.6
1                 17.1                 23.6                 12.3
2                 16.8                 27.2                  8.8
3                 14.6                 19.9                 11.1
4                 13.4                 19.1                  8.0

   Atmospheric pressure  Average humidity  Total rainfall  Average
visibility  \
0                   0.0                56            1.52
10.0
1                   0.0                63            0.00
10.0
2                   0.0                47            0.25
10.0
3                   0.0                49            0.00
10.0
4                   0.0                65            0.00
10.0

   Average wind speed  Maximum wind speed  Year  Month  Day
```

```
0                 26.9                    53.5  2014       1    1
1                 14.4                    27.8  2014       1    2
2                 31.9                    61.1  2014       1    5
3                 27.0                    38.9  2014       1    6
4                 20.6                    35.2  2014       1    7

#We have dropped the first column

data = pd.DataFrame(data)

data.head()

         0         1         2       3         4         5         6  \
0  0.371338  0.000000  0.210224  0.1750  0.316901  0.276119  0.391885

1  0.370707  0.022712  0.230700  0.1500  0.323944  0.283582  0.385115

2  0.369697  0.038830  0.303092  0.3625  0.391549  0.355224  0.471577

3  0.369697  0.046706  0.322727  0.2875  0.408451  0.365672  0.442308

4  0.368308  0.073570  0.305960  0.4125  0.518310  0.489552  0.492115


         7         8         9   10        11        12        13
14  \
0  0.543662  0.577011  0.478689  0.0  0.577320  0.084304  0.019531
0.547862
1  0.481690  0.542529  0.468852  0.0  0.649485  0.000000  0.019531
0.293279
2  0.473239  0.625287  0.354098  0.0  0.484536  0.013866  0.019531
0.649695
3  0.411268  0.457471  0.429508  0.0  0.505155  0.000000  0.019531
0.549898
4  0.377465  0.439080  0.327869  0.0  0.670103  0.000000  0.019531
0.419552

         15   16   17        18
0  0.640719  0.0  0.0  0.000000
1  0.332934  0.0  0.0  0.033333
2  0.731737  0.0  0.0  0.133333
3  0.465868  0.0  0.0  0.166667
4  0.421557  0.0  0.0  0.200000

data.shape

(1382, 19)

data.isnull().sum()
```

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
18    0
dtype: int64
```

```python
#No null columns

corr_mat = data.corr()
plt.figure(figsize=(20,20))
sns.heatmap(corr_mat, annot = True, linewidth = 1, linecolor =
'black')
plt.show()
```

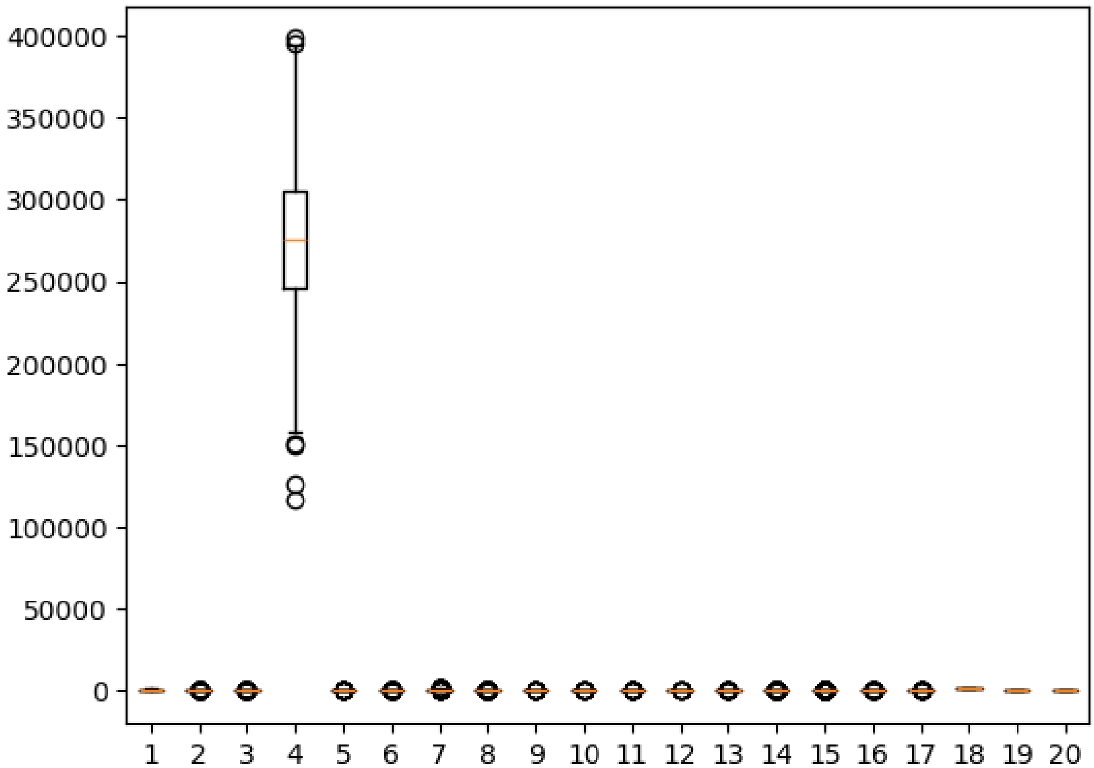



```
data['Optimality'] = 10
low = data['Energy Consumption'].quantile(0.05)
high = data['Energy Consumption'].quantile(0.55)
data.loc[(data['Energy Consumption']>=low) & (data['Energy
Consumption']<=high), 'Optimality'] = 1
data.loc[~((data['Energy Consumption']>=low) & (data['Energy
Consumption']<=high)), 'Optimality'] = 0

data.head()
```

```
print(low)
print(high)
```

```
198111.05
281782.25
```

```
plt.boxplot(data)
plt.show()
```




```
from sklearn.model_selection import train_test_split

#Selecting Features
x = data.drop([ 'Year', 'Month', 'Day', 'Optimality', ], axis=1)
y = data['Optimality']

x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=42, test_size=0.2)

from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
x_train_sc = sc.fit_transform(x_train)
x_test_sc = sc.fit_transform(x_test)

from sklearn.metrics import f1_score
```

```python
#Sigmoid Function
def sigmoid(z):
    return 1/(1+np.exp(-z))

#Cost Function (Based on log-loss)
def Cost_Function(x, y, beta):
    m = len(y)
    h = sigmoid(x @ beta)
    J = -1/m * np.sum(y * np.log(h) + (1-y) * np.log(1-h))
    return J

#Gradient Descent
def gradient_descent(x, y, beta, alpha, iters):
    m = len(y)
    for i in range(iters):
        h = sigmoid(np.dot(x, beta))
        beta = beta - alpha * 1/m * (x.T @ (h - y))
    return beta

#Prediction Function
def predict(x, beta):
    probability = sigmoid(x @ beta)
    return [1 if prob >= 0.5 else 0 for prob in probability]

#Logistic Regression
def logistic_regression(x_train_sc, x_test_sc, y_train, y_test,
alpha=0.1, iters=1000):

    x_train_sc = np.c_[np.ones((x_train_sc.shape[0], 1)), x_train_sc]

    x_test_sc = np.c_[np.ones((x_test_sc.shape[0], 1)), x_test_sc]

    print("Shape of x_train_scaled:", x_train_sc.shape)
    print("Shape of x_test_scaled:", x_test_sc.shape)

    # Initialize beta to zero
    beta = np.zeros(x_train_sc.shape[1])
    print("Shape of beta:", beta.shape)

    # Train the model
    weight = gradient_descent(x_train_sc, y_train, beta, alpha, iters)

    # Make predictions on the test set
    predictions = predict(x_test_sc, weight)

    return weight, predictions

weight, predictions = logistic_regression(x_train_sc, x_test_sc,
y_train, y_test)
print("\nWeights:", weight)
```

```
# Model Evaluation
accuracy_scratch = np.mean(predictions == y_test)
accuracy_scratch
F1_scratch = f1_score(y_test, predictions)
F1_scratch
```

```
Shape of x_train_scaled: (1105, 18)
Shape of x_test_scaled: (277, 18)
Shape of beta: (18,)
```

```
Weights: [ 0.4453377  -0.6165742  -0.23130351 -0.13757097 -2.93782944
0.33700166
  0.58019296  0.22046395  0.39704644  0.65456609  0.83166099
0.52282195
  0.01062281  0.02225452  0.07821289  0.06448857  0.10463371  0.161629
]
```

```
0.7846153846153846
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
logr= LogisticRegression()

logr.fit(x_train_sc, y_train)

y_pred_logr = logr.predict(x_test_sc)
accuracy_logr = accuracy_score(y_test, y_pred_logr)
accuracy_logr
F1_logr = f1_score(y_test, y_pred_logr)
F1_logr
```

```
0.8384615384615385
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
knn= KNeighborsClassifier(n_neighbors = 10)

knn.fit(x_train_sc, y_train)

y_pred_knn = knn.predict(x_test_sc)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
accuracy_knn
F1_knn = f1_score(y_test, y_pred_knn)
F1_knn
```

```
0.8045977011494253
```

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(random_state=42)
```

```
dtc.fit(x_train_sc, y_train)

y_predicted = dtc.predict(x_test_sc)

accuracy_dtc = accuracy_score(y_test, y_predicted)
accuracy_dtc
F1_dtc = f1_score(y_test, y_predicted)
F1_dtc
```

```
0.8957528957528957
```

```
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

#Training the model
rf_classifier.fit(x_train_sc, y_train)

#Predictions
y_pred_rf = rf_classifier.predict(x_test_sc)

#Model Evaluation
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest Classifier- \nAccuracy:", accuracy_rf)
F1_rf = f1_score(y_test, y_pred_rf)
print(f"F1 Score: {F1_rf:.2f}")
```

```
Random Forest Classifier-
Accuracy: 0.8989169675090253
F1 Score: 0.89
```

```
from sklearn.svm import SVC
svc_classifier = SVC(kernel='rbf')

#Training the model
svc_classifier.fit(x_train_sc, y_train)

# Make predictions
y_pred_svc = svc_classifier.predict(x_test_sc)

#Model Evaluation
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print("Support Vector Classifier- \nAccuracy:", accuracy_svc)
F1_svc = f1_score(y_test, y_pred_svc)
print(f"F1 Score: {F1_svc:.2f}")
```

```
Support Vector Classifier-
Accuracy: 0.8989169675090253
F1 Score: 0.90
```

```
from sklearn.metrics import confusion_matrix
confusion_scratch = confusion_matrix(y_test, predictions)
```

```python
confusion_sklearn = confusion_matrix(y_test, y_pred_logr)
confusion_knn = confusion_matrix(y_test, y_pred_knn)
confusion_dt = confusion_matrix(y_test, y_pred_dtc)
confusion_rf = confusion_matrix(y_test, y_pred_rf)
confusion_svm = confusion_matrix(y_test, y_pred_svc)
df = {
    'Metrics' : ['Accuracy', 'F1-Score','Confusion Matrix'],
    'LR Scratch' : [accuracy_scratch , F1_scratch, confusion_scratch],
    'LR sklearn' : [accuracy_sklearn, F1_sklearn, confusion_sklearn],
    'KNN' : [accuracy_knn, F1_knn, confusion_knn],
    'Decision Tree' : [accuracy_dtc, F1_dtc, confusion_dt],
    'Random Forest' : [accuracy_rf, F1_rf, confusion_rf],
    'SVM' : [accuracy_svc, F1_svc, confusion_svm]
}
df = pd.DataFrame(df)
df = df.set_index('Metrics')
df
```

Metrics	LR Scratch	LR sklearn
Accuracy	0.797834	0.848375
F1-Score	0.784615	0.838462
Confusion Matrix	[[119, 18], [38, 102]]	[[126, 11], [31, 109]]

Metrics	KNN	Decision Tree
Accuracy	0.815884	0.902527
F1-Score	0.804598	0.895753
Confusion Matrix	[[121, 16], [35, 105]]	[[134, 3], [24, 116]]

Metrics	Random Forest	SVM
Accuracy	0.898917	0.898917
F1-Score	0.892308	0.9
Confusion Matrix	[[133, 4], [24, 116]]	[[123, 14], [14, 126]]

```python
from sklearn.model_selection import GridSearchCV
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

grid_search_dt =
GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
param_grid=param_grid_dt, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)
grid_search_dt.fit(x_train_sc, y_train)
```

```
best_params_dt = grid_search_dt.best_params_
print("Best Hyperparameters for Decision Tree:", best_params_dt)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Hyperparameters for Decision Tree: {'criterion': 'gini',
'max_depth': None, 'max_features': None, 'min_samples_leaf': 1,
'min_samples_split': 2}
```

```
#KNN
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

grid_search_knn = GridSearchCV(estimator=KNeighborsClassifier(),
param_grid=param_grid_knn, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)
grid_search_knn.fit(x_train_sc, y_train)
best_params_knn = grid_search_knn.best_params_
print("Best Hyperparameters for KNN:", best_params_knn)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Hyperparameters for KNN: {'metric': 'manhattan', 'n_neighbors':
11, 'weights': 'uniform'}
```

```
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

grid_search_dt =
GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
param_grid=param_grid_dt, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)
grid_search_dt.fit(x_train_sc, y_train)
best_params_dt = grid_search_dt.best_params_
print("Best Hyperparameters for Decision Tree:", best_params_dt)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Hyperparameters for Decision Tree: {'criterion': 'gini',
'max_depth': None, 'max_features': None, 'min_samples_leaf': 1,
'min_samples_split': 2}
```

```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}

grid_search_rf =
GridSearchCV(estimator=RandomForestClassifier(random_state=42),
param_grid=param_grid_rf, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)
grid_search_rf.fit(x_train_sc, y_train)
best_params_rf = grid_search_rf.best_params_
print("Best Hyperparameters for Random Forest:", best_params_rf)
```

```
Fitting 5 folds for each of 648 candidates, totalling 3240 fits
```

```
#SVC
param_grid_svc = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 0.01, 0.1, 1],
    'degree': [2, 3, 4],
    'class_weight': [None, 'balanced']
}

grid_search_svc = GridSearchCV(estimator=SVC(random_state=42),
param_grid=param_grid_svc, cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)
grid_search_svc.fit(x_train_scaled, y_train)
best_params_svc = grid_search_svc.best_params_
print("Best Hyperparameters for SVC:", best_params_svc)
```