In [1]:

```
# !pip install tensorflow
```

In [2]:

```
import pandas as pd
import numpy as np


from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

import pickle
```

In [3]:

```
df1  = pd.read_csv('Absenteeism_preprocessed.csv')
df1.head()
```

Out[3]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month of absence	Day of the week	Transportation expense	Distance from Residence to Work	Work load Average/day
0	0	0	0	1	7	3	289	36	239554
1	0	0	0	0	7	3	118	13	239554
2	0	0	0	1	7	4	179	51	239554
3	1	0	0	0	7	5	279	5	239554
4	0	0	0	1	7	5	289	36	239554

In [4]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 655 entries, 0 to 654
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Reason_1                         655 non-null    int64
 1   Reason_2                         655 non-null    int64
 2   Reason_3                         655 non-null    int64
 3   Reason_4                         655 non-null    int64
 4   Month of absence                 655 non-null    int64
 5   Day of the week                  655 non-null    int64
 6   Transportation expense           655 non-null    int64
 7   Distance from Residence to Work  655 non-null    int64
 8   Work load Average/day            655 non-null    int64
 9   Education                        655 non-null    int64
 10  Son                              655 non-null    int64
 11  Pet                              655 non-null    int64
 12  Body mass index                  655 non-null    int64
 13  Excessive Absentise              655 non-null    int64
dtypes: int64(14)
memory usage: 71.8 KB
```

In [5]:
```
target = df1.iloc[:,-1]
target
```

Out[5]:
```
0      1
1      0
2      0
3      1
4      0
      ..
650    1
651    1
652    0
653    0
654    0
Name: Excessive Absentise, Length: 655, dtype: int64
```

In [6]:
```
features = df1.iloc[:,:-1]
```

In [7]:
```
df5 = features.copy()
```

Scaling the features except for Reasons and Education column

In [8]:
```
columns_to_scale =  ['Month of absence','Day of the week', 'Transportation expense'
                     'Distance from Residence to Work', 'Work load Average/day ',
                     'Son', 'Pet', 'Body mass index']
```

In [9]:
```
scaler = StandardScaler()
```

In [10]:
```
scaler.fit(df5[columns_to_scale])
```

Out[10]:
```
▾ StandardScaler
StandardScaler()
```

In [11]:
```
df5[columns_to_scale] = scaler.transform(df5[columns_to_scale])
```

In [12]:
```
df5
```

Out[12]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month of absence	Day of the week	Transportation expense	Distance from Residence to Work	Wo Aver
0	0	0	0	1	0.241620	-0.626495	1.041496	0.431831	-0
1	0	0	0	0	0.241620	-0.626495	-1.580091	-1.135012	-0
2	0	0	0	1	0.241620	0.073768	-0.644905	1.453685	-0
3	1	0	0	0	0.241620	0.774031	0.888187	-1.680000	-0
4	0	0	0	1	0.241620	0.774031	1.041496	0.431831	-0
...	...	...	...	...	...	...	...	...	
650	1	0	0	0	0.241620	-0.626495	1.041496	0.431831	-0
651	1	0	0	0	0.241620	-0.626495	0.213626	-1.271259	-0
652	0	0	0	0	-1.836858	-0.626495	-1.580091	-1.066888	0
653	0	0	0	0	-1.836858	0.073768	0.152303	0.363707	0
654	0	0	0	0	-1.836858	1.474294	-0.644905	1.044943	0

655 rows × 13 columns

In [13]:
```
x_train, x_test, y_train, y_test = train_test_split(df5,target,train_size=0.8, rand
```

In [14]:
```
print(f'x_train size = {x_train.shape}')
print(f'x_train size = {x_test.shape}')
print(f'x_train size = {y_train.shape}')
print(f'x_train size = {y_test.shape}')
```

```
x_train size = (524, 13)
x_train size = (131, 13)
x_train size = (524,)
x_train size = (131,)
```

Training the model with Logestic Regression

In [15]:
```
reg = LogisticRegression()
```

In [16]:
```
reg.fit(x_train, y_train)
```

Out[16]:
```
▾ LogisticRegression
LogisticRegression()
```

In [17]:
```
reg.score(x_train, y_train)
```

Out[17]:
```
0.7366412213740458
```

Manual Checking the outputs

In [18]:
```
model_outputs = reg.predict(x_train)
```

In [19]:
```
(model_outputs == y_train).sum() / y_train.shape[0]
```

Out[19]:
```
0.7366412213740458
```

What are the values of Intercept(bias) and coefficients(weights)

In [20]:
```
reg.intercept_[0]
```

Out[20]:
```
-1.3475563128218284
```

In [21]:
```
reg.coef_[0]
```

Out[21]:
```
array([ 2.35444722,  0.42424739,  2.45235337,  0.59205429,  0.09672736,
       -0.25793199,  0.45874475,  0.01184495, -0.05925763,  0.21126837,
        0.59530394, -0.22507848,  0.14383312])
```

In [22]:
```
feature_names = df5.columns.values
```

In [23]:
```
np.transpose(reg.coef_)
```

Out[23]:
```
array([[ 2.35444722],
       [ 0.42424739],
       [ 2.45235337],
       [ 0.59205429],
       [ 0.09672736],
       [-0.25793199],
       [ 0.45874475],
       [ 0.01184495],
       [-0.05925763],
       [ 0.21126837],
       [ 0.59530394],
       [-0.22507848],
       [ 0.14383312]])
```

Designing a summary table of the values we got

In [24]:
```
summary_table = pd.DataFrame(columns=['feature_names'], data = feature_names )
```

In [25]:
```
summary_table['Coefficients'] = np.transpose(reg.coef_)
summary_table.index = summary_table.index+1
```

In [26]:
```
summary_table.loc[0] = ['Intercept', reg.intercept_[0]]
summary_table
```

Out[26]:

	feature_names	Coefficients
1	Reason_1	2.354447
2	Reason_2	0.424247
3	Reason_3	2.452353
4	Reason_4	0.592054
5	Month of absence	0.096727
6	Day of the week	-0.257932
7	Transportation expense	0.458745
8	Distance from Residence to Work	0.011845
9	Work load Average/day	-0.059258
10	Education	0.211268
11	Son	0.595304
12	Pet	-0.225078
13	Body mass index	0.143833
0	Intercept	-1.347556

In [27]:

```
summary_table['Odds_data'] = np.exp(summary_table['Coefficients'])
summary_table.sort_values(by = 'Odds_data', ascending= False, inplace=True)
summary_table
```

Out[27]:

	feature_names	Coefficients	Odds_data
3	Reason_3	2.452353	11.615650
1	Reason_1	2.354447	10.532305
11	Son	0.595304	1.813582
4	Reason_4	0.592054	1.807698
7	Transportation expense	0.458745	1.582087
2	Reason_2	0.424247	1.528440
10	Education	0.211268	1.235244
13	Body mass index	0.143833	1.154691
5	Month of absence	0.096727	1.101560
8	Distance from Residence to Work	0.011845	1.011915
9	Work load Average/day	-0.059258	0.942464
12	Pet	-0.225078	0.798454
6	Day of the week	-0.257932	0.772648
0	Intercept	-1.347556	0.259875

In [28]:

```

with open('Scaler','wb') as file:
    pickle.dump(scaler,file)
```

In [29]:

```
with open('Model','wb') as file:
    pickle.dump(reg, file)
```

In [ ]:

In [30]:
```
reg.predict_proba(x_test)[:,1]
```

Out[30]:
```
array([0.21392488, 0.78261124, 0.2253243 , 0.54617877, 0.29693789,
       0.34013367, 0.2774964 , 0.27061777, 0.83753207, 0.68944643,
       0.55925977, 0.75645576, 0.553982  , 0.17687561, 0.54487847,
       0.29917156, 0.46750321, 0.78607212, 0.53924195, 0.58934368,
       0.17636083, 0.27329505, 0.68836919, 0.62205449, 0.95266802,
       0.26245265, 0.18945113, 0.94040087, 0.10798627, 0.84648491,
       0.6700766 , 0.39598874, 0.27329505, 0.38632525, 0.19238617,
       0.32537085, 0.82090642, 0.54823791, 0.83396485, 0.87518863,
       0.16247325, 0.54381635, 0.16262248, 0.47229121, 0.80306141,
       0.75577829, 0.71612269, 0.15929913, 0.34953905, 0.204146  ,
       0.87822067, 0.60523219, 0.42268436, 0.17035396, 0.15981807,
       0.14236069, 0.24432427, 0.84441236, 0.23367267, 0.84576359,
       0.68944643, 0.85543627, 0.27061777, 0.09539072, 0.20747703,
       0.20542426, 0.32185929, 0.31909864, 0.16446278, 0.1516368 ,
       0.27228469, 0.46360239, 0.7152524 , 0.860638  , 0.42692132,
       0.52263817, 0.1260849 , 0.80519696, 0.60424489, 0.17656561,
       0.41048113, 0.89586642, 0.3012483 , 0.33054246, 0.86479266,
       0.79043293, 0.29458032, 0.125194  , 0.11156181, 0.44716334,
       0.86388617, 0.48051059, 0.33054246, 0.76644627, 0.57160518,
       0.27653291, 0.18260829, 0.28394475, 0.57989304, 0.68231944,
       0.38078707, 0.18945113, 0.89699149, 0.23017376, 0.31801396,
       0.2617089 , 0.22461474, 0.68558641, 0.85543627, 0.11569973,
       0.60165255, 0.28703714, 0.64746951, 0.17207546, 0.19993927,
       0.31801396, 0.17460531, 0.94843471, 0.95473842, 0.53111433,
       0.87234635, 0.76284803, 0.56224826, 0.19313475, 0.30346477,
       0.93431851, 0.71192601, 0.25476862, 0.39870535, 0.1534825 ,
       0.23064868])
```

In [31]:
```
reg.predict(x_test)
```

Out[31]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0],
      dtype=int64)
```

In [32]:
```
reg.score(x_test,y_test)
```

Out[32]:
```
0.732824427480916
```

In [33]:
```
pd.DataFrame(list(reg.get_params().items()),columns=['Parameter','Values'])
```

Out[33]:

	Parameter	Values
0	C	1.0
1	class_weight	None
2	dual	False
3	fit_intercept	True
4	intercept_scaling	1
5	l1_ratio	None
6	max_iter	100
7	multi_class	auto
8	n_jobs	None
9	penalty	l2
10	random_state	None
11	solver	lbfgs
12	tol	0.0001
13	verbose	0
14	warm_start	False

In [34]:
```
reg.get_params().items()
```

Out[34]:
```
dict_items([('C', 1.0), ('class_weight', None), ('dual', False), ('fit_intercept',
True), ('intercept_scaling', 1), ('l1_ratio', None), ('max_iter', 100), ('multi_cl
ass', 'auto'), ('n_jobs', None), ('penalty', 'l2'), ('random_state', None), ('solv
er', 'lbfgs'), ('tol', 0.0001), ('verbose', 0), ('warm_start', False)])
```

In [35]:
```
reg.get_params()
```

Out[35]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [36]:

```
# DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier

# Instantiate the Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=20)

# Train the model using the training data
dt.fit(x_train, y_train)

# Print the accuracy score on the training data
dt_training_accuracy = dt.score(x_train, y_train)
print(f'Decision Tree Training accuracy: {dt_training_accuracy:.2f}')

# Evaluate the model on the test data
dt_test_accuracy = dt.score(x_test, y_test)
print(f'Decision Tree Test accuracy: {dt_test_accuracy:.2f}')
```

```
Decision Tree Training accuracy: 0.98
Decision Tree Test accuracy: 0.70
```

In [37]:

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest Classifier
rf = RandomForestClassifier(random_state=20)

# Train the model using the training data
rf.fit(x_train, y_train)

# Print the accuracy score on the training data
rf_training_accuracy = rf.score(x_train, y_train)
print(f'Random Forest Training accuracy: {rf_training_accuracy:.2f}')

# Evaluate the model on the test data
rf_test_accuracy = rf.score(x_test, y_test)
print(f'Random Forest Test accuracy: {rf_test_accuracy:.2f}')


# Best model RandomForestClassifier
```

```
Random Forest Training accuracy: 0.98
Random Forest Test accuracy: 0.78
```

In [40]:

```
from sklearn.svm import SVC

# Instantiate the Support Vector Classifier
svm = SVC(random_state=20)

# Train the model using the training data
svm.fit(x_train, y_train)

# Print the accuracy score on the training data
svm_training_accuracy = svm.score(x_train, y_train)
print(f'SVM Training accuracy: {svm_training_accuracy:.2f}')

# Evaluate the model on the test data
svm_test_accuracy = svm.score(x_test, y_test)
print(f'SVM Test accuracy: {svm_test_accuracy:.2f}')
```

```
SVM Training accuracy: 0.77
SVM Test accuracy: 0.73
```

In [41]:

```
from sklearn.neighbors import KNeighborsClassifier

# Instantiate the K-Nearest Neighbors Classifier
knn = KNeighborsClassifier()

# Train the model using the training data
knn.fit(x_train, y_train)

# Print the accuracy score on the training data
knn_training_accuracy = knn.score(x_train, y_train)
print(f'KNN Training accuracy: {knn_training_accuracy:.2f}')

# Evaluate the model on the test data
knn_test_accuracy = knn.score(x_test, y_test)
print(f'KNN Test accuracy: {knn_test_accuracy:.2f}')
```

```
KNN Training accuracy: 0.78
KNN Test accuracy: 0.70
```

In [38]:

```
with open('Model_rf','wb') as file: #saving the Random forest model parameters
    pickle.dump(rf, file)
```

In [39]:

```
rf_test_accuracy # We choose the Random Forest model whose test accuracy is highest
```

Out[39]:

```
0.7786259541984732
```

In [ ]:

In [ ]: