In [12]:
```
import pandas as pd
import numpy as np
data = pd.read_excel('weight.xlsx')
data
```

Out[12]:

	Sample ID	parameter1	parameter2	parameter3	parameter4	parameter5	parameter6	paramet
0	5668	-0.824167	-24.990002	-11.643333	143.503006	343.649994	803	1.4382
1	6074	0.592500	-22.719999	-9.906667	158.704010	561.040039	184	0.8206
2	6118	-0.611667	-25.520000	-11.480000	146.677002	412.039947	519	0.1281
3	6259	1.749167	-22.200001	-8.890001	162.697006	504.680023	214	0.1354
4	6363	0.719167	-22.799999	-9.480000	158.247009	580.250000	292	0.4229
...	...	...	...	...	...	...	...	
1259	26453	19.747499	9.990000	16.459999	357.351990	1853.469971	238	0.9113
1260	26454	19.747499	9.990000	16.459999	357.351990	1853.469971	238	0.9113
1261	26455	19.747499	9.990000	16.459999	357.351990	1853.469971	238	0.9113
1262	26456	19.747499	9.990000	16.459999	357.351990	1853.469971	238	0.9113

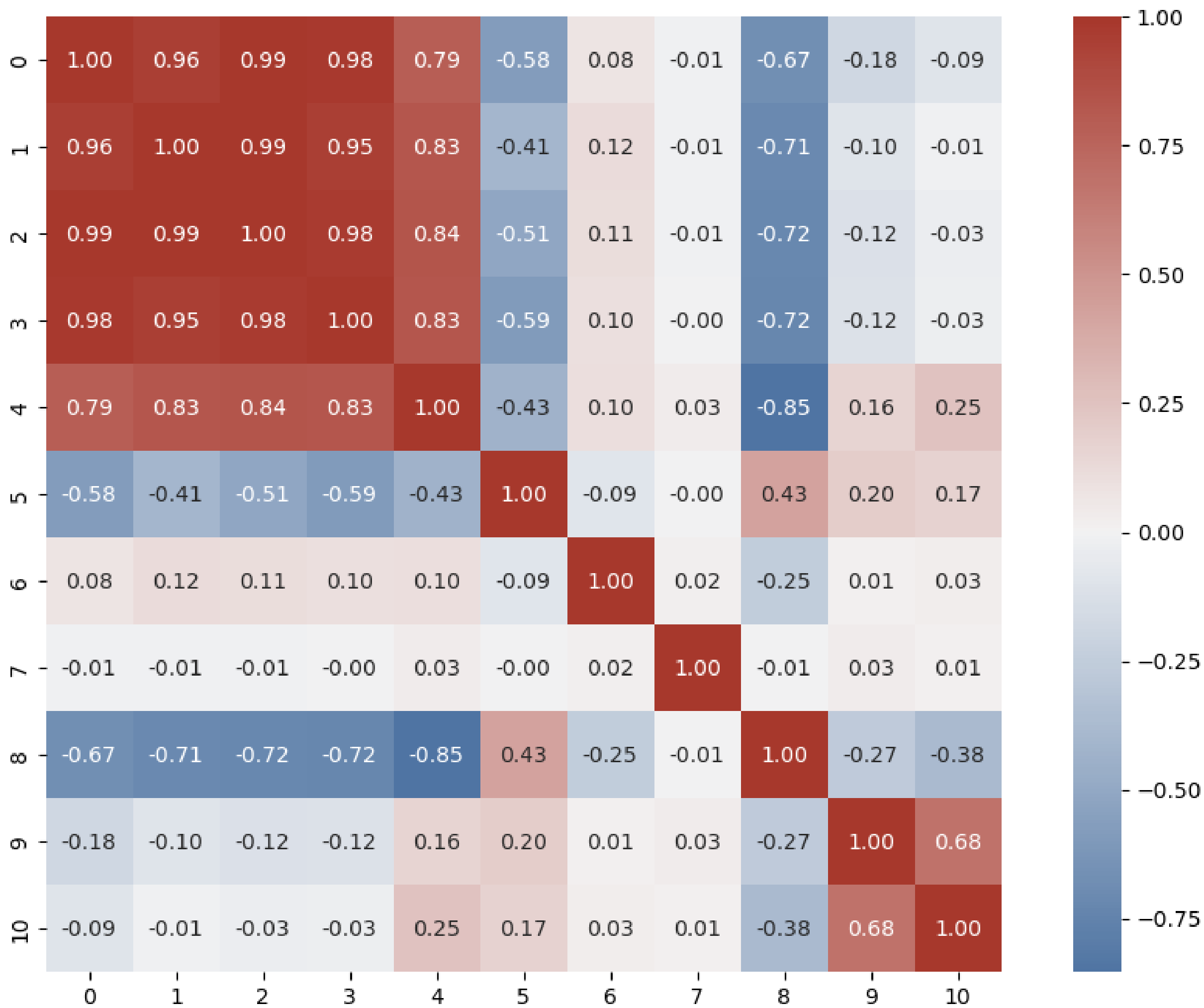
In [13]:
```
# Preprocessing
# Preprocessing
# Delete useless data
data.drop(["Sample ID"],axis=1,inplace=True)
# Delete duplicate data
data.drop_duplicates(inplace=True)
# delete missing data
data=data.dropna()
```

In [14]:
```
# Characteristic standardisation
y=data["level of suitability"]
data.drop("level of suitability",axis=1,inplace=True)
X=data
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit(X).transform(X)
```

```
C:\Users\ZSY\AppData\Local\Temp\ipykernel_20100\2541281730.py:3: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data.drop("level of suitability",axis=1,inplace=True)
```

In [15]:
```
# Visualisation of indicators correlation
import matplotlib.pyplot as plt
import seaborn as sns
matrix=pd.DataFrame(X).corr()
cmap = sns.diverging_palette(250, 15, s=75, l=40, n=9, center=”light”, as_cmap=True)
plt.figure(figsize=(12, 8))
sns.heatmap(matrix,  center=0, annot=True,fmt='.2f', square=True, cmap=cmap)
```

Out[15]: <AxesSubplot:>



In [16]:
```
# Model selection and training
# Train dataset and test dataset division
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=10)
```

In [17]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
cls = LogisticRegression()
cls.fit(X_train, y_train)
print(cls.coef_[0])
LR_pred=cls.predict(X_train)
LR_test_pred=cls.predict(X_test)
```

```
[-0.82156316 -1.62070392 -0.43521202 -1.67924567 -1.41712739 -0.70184061
  0.02703885 -0.19287723  0.90738605 -0.29850257 -0.66295556]
```

In [18]:
```
# Support Vector Machines
from sklearn.svm import SVC
cls = SVC()
cls.fit(X_train, y_train)
SVC_pred=cls.predict(X_train)
SVC_test_pred=cls.predict(X_test)
```

In [19]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
cls = RandomForestClassifier(max_depth=50, max_features=2, min_samples_leaf=1,
                       min_samples_split=7, n_estimators=80,random_state=0)
cls.fit(X_train, y_train)
print(cls.feature_importances_)
RF_pred=cls.predict(X_train)
RF_test_pred=cls.predict(X_test)
```

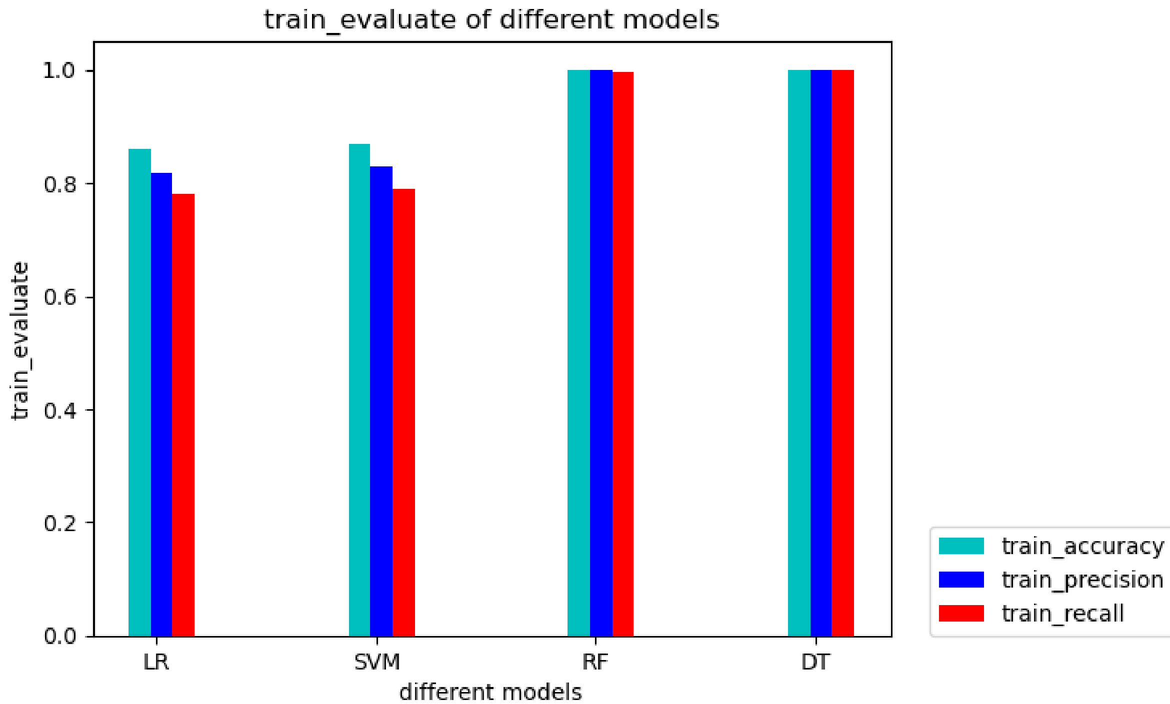
```
[0.1173547  0.16074526 0.14067698 0.14447819 0.15297809 0.05662247
 0.03195019 0.01650018 0.08795861 0.04083468 0.04990066]
```

In [20]:
```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
cls = DecisionTreeClassifier()
cls.fit(X_train, y_train)
print(cls.feature_importances_)
DT_pred=cls.predict(X_train)
DT_test_pred=cls.predict(X_test)
```

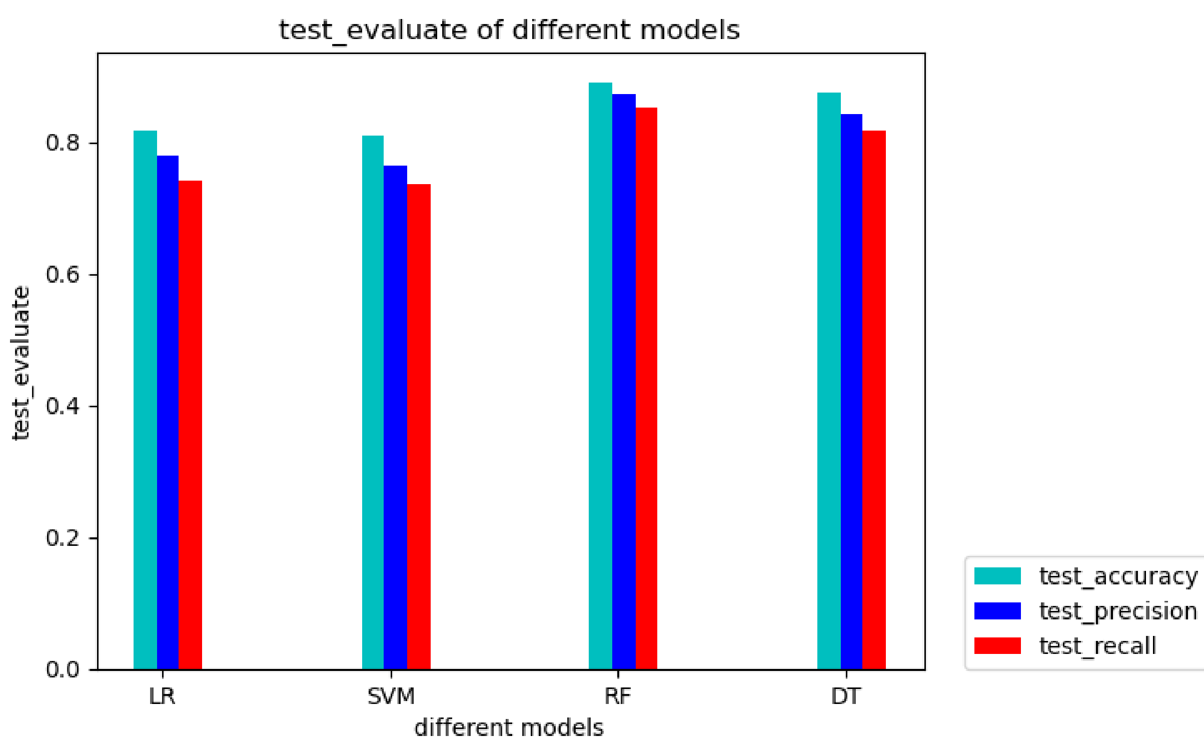
```
[0.08448237 0.55138008 0.00817953 0.04467668 0.17650061 0.03793559
 0.03245952 0.01947661 0.01498965 0.01901294 0.01090641]
```

In [21]:

```
# Model Evaluation
from sklearn.metrics import accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import numpy as np
#Train dataset
LR_train_acc=accuracy_score(y_train, LR_pred)
SVC_train_acc=accuracy_score(y_train, SVC_pred)
RF_train_acc=accuracy_score(y_train, RF_pred)
DT_train_acc=accuracy_score(y_train, DT_pred)

LR_train_pre=precision_score(y_train, LR_pred, average='macro')
SVC_train_pre=precision_score(y_train, SVC_pred, average='macro')
RF_train_pre=precision_score(y_train, RF_pred, average='macro')
DT_train_pre=precision_score(y_train, DT_pred, average='macro')

LR_train_rec=recall_score(y_train, LR_pred, average='macro')
SVC_train_rec=recall_score(y_train, SVC_pred, average='macro')
RF_train_rec=recall_score(y_train, RF_pred, average='macro')
DT_train_rec=recall_score(y_train, DT_pred, average='macro')
# Mapping
plt.figure()
bar_width=0.1
x = np.arange(4)
x_tick=["LR","SVM","RF","DT"]
acc = np.array([LR_train_acc, SVC_train_acc, RF_train_acc, DT_train_acc])
pre = np.array([LR_train_pre, SVC_train_pre, RF_train_pre, DT_train_pre])
rec = np.array([LR_train_rec, SVC_train_rec, RF_train_rec, DT_train_rec])
plt.bar(x, acc, bar_width, align="center", color="c", label="train_accuracy")
plt.bar(x+bar_width, pre, bar_width, align="center", color="b", label="train_precision")
plt.bar(x+2*bar_width, rec, bar_width, align="center", color="r", label="train_recall")
plt.xticks(x+bar_width*3/4, x_tick)
plt.xlabel("different models")
plt.ylabel("train_evaluate")
plt.title("train_evaluate of different models")
plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
plt.show()
```


train_evaluate of different models
train_evaluate
1.0
0.8
0.6
0.4
0.2
0.0
LR
SVM
RF
DT
different models
train_accuracy
train_precision
train_recall

In [22]:
```
#Test dataset
LR_test_acc=accuracy_score(y_test, LR_test_pred)
SVC_test_acc=accuracy_score(y_test, SVC_test_pred)
RF_test_acc=accuracy_score(y_test, RF_test_pred)
DT_test_acc=accuracy_score(y_test, DT_test_pred)

LR_test_pre=precision_score(y_test, LR_test_pred, average='macro')
SVC_test_pre=precision_score(y_test, SVC_test_pred, average='macro')
RF_test_pre=precision_score(y_test, RF_test_pred, average='macro')
DT_test_pre=precision_score(y_test, DT_test_pred, average='macro')

LR_test_rec=recall_score(y_test, LR_test_pred, average='macro')
SVC_test_rec=recall_score(y_test, SVC_test_pred, average='macro')
RF_test_rec=recall_score(y_test, RF_test_pred, average='macro')
DT_test_rec=recall_score(y_test, DT_test_pred, average='macro')
# Mapping
plt.figure()
bar_width=0.1
x = np.arange(4)
x_tick=["LR","SVM","RF","DT"]
acc = np.array([LR_test_acc, SVC_test_acc, RF_test_acc, DT_test_acc])
pre = np.array([LR_test_pre, SVC_test_pre, RF_test_pre, DT_test_pre])
rec = np.array([LR_test_rec, SVC_test_rec, RF_test_rec, DT_test_rec])
plt.bar(x, acc, bar_width, align="center", color="c", label="test_accuracy")
plt.bar(x+bar_width, pre, bar_width, align="center", color="b", label="test_precision")
plt.bar(x+2*bar_width, rec, bar_width, align="center", color="r", label="test_recall")
plt.xticks(x+bar_width*3/4, x_tick)
plt.xlabel("different models")
plt.ylabel("test_evaluate")
plt.title("test_evaluate of different models")
plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
plt.show()
```



In [ ]:

```

```