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

```
df=sns.load_dataset('tips')
df.head(4)
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2

```
df.day.value_counts()
```

day	count
Sat	87
Sun	76
Thur	62
Fri	19

dtype: int64

```
df.isnull().sum()
```

	0
total_bill	0
tip	0
sex	0
smoker	0
day	0
time	0
size	0

```
df.day.unique()
```

```
['Sun', 'Sat', 'Thur', 'Fri']
Categories (4, object): ['Thur', 'Fri', 'Sat', 'Sun']
```

```
df.time.unique()
```

```
['Dinner', 'Lunch']
Categories (2, object): ['Lunch', 'Dinner']
```

```
df.describe()
```

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

```
df.shape
```

```
(244, 7)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

```
df.columns
```

```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'], dtype='object')
```

```
#convert the catagorical value into the numerical value using the labelencoder
from sklearn.preprocessing import LabelEncoder
encode=LabelEncoder()
df['time']=encode.fit_transform(df['time'])#here the time is the dependent varable
```

```
df.time.unique()
```

```
array([0, 1])
```

```
#dependent and independent features
X=df.drop('time',axis=1)
y=df.time
```

```
X.head(4)
```

	total_bill	tip	sex	smoker	day	size
0	16.99	1.01	Female	No	Sun	2
1	10.34	1.66	Male	No	Sun	3
2	21.01	3.50	Male	No	Sun	3
3	23.68	3.31	Male	No	Sun	2

```
y.head(4)
```

	time
0	0
1	0
2	0
3	0

dtype: int64

```
#train_test_split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=42)
```

```
X_train
```

	total_bill	tip	sex	smoker	day	size
234	15.53	3.00	Male	Yes	Sat	2
227	20.45	3.00	Male	No	Sat	4
180	34.65	3.68	Male	Yes	Sun	4
5	25.29	4.71	Male	No	Sun	4
56	38.01	3.00	Male	Yes	Sat	4
...	...	...	...	...	...	...
106	20.49	4.06	Male	Yes	Sat	2
14	14.83	3.02	Female	No	Sun	2
92	5.75	1.00	Female	Yes	Fri	2
179	34.63	3.55	Male	Yes	Sun	2
102	44.30	2.50	Female	Yes	Sat	3

170 rows × 6 columns

```
#use pipeline for automation
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer #for handling the missing value
from sklearn.preprocessing import OneHotEncoder #catagorical to numerical
from sklearn.compose import ColumnTransformer #combine pipelines
from sklearn.preprocessing import StandardScaler #fearure engineering
```

```
categorical_cols=['sex','smoker','day']
numerical_cols= ['total_bill','tip','size']
```

```


#feature engineering automation
#numerical pipeline
num_pipeline=Pipeline(
    steps=[('imputer',SimpleImputer(strategy='median')),## handling missing values
           ('scaler',StandardScaler())##feature scaling
          ]
)

#categorical pipeline
cat_pipeline=Pipeline(
    steps=[('imputer',SimpleImputer(strategy='most_frequent')),## handling null values
          ('onehotencoder',OneHotEncoder()) ##categorical to numerical
          ]
)
```

```
#combine pipeline using columntransformer
preprocessor=ColumnTransformer([
    ('num_pipeline',num_pipeline,numerical_cols),
    ('cat_pipeline',cat_pipeline,categorical_cols)
])
```

```
X_train=preprocessor.fit_transform(X_train)
X_test=preprocessor.transform(X_test)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```
#model training automation
models={
    'random_forest': RandomForestClassifier(),
    'logistic_regression': LogisticRegression()
}
```

```
from sklearn.metrics import accuracy_score
```

```
def evaluate_model(X_train,X_test,y_train,y_test,models):
    report={}
    for i in range(len(models)):
```

```
        model=list(models.values())[i]
        #training model
        model.fit(X_train,y_train)

        #predict testing data
        y_pred=model.predict(X_test)

        #accuracy score
        test_model_score=accuracy_score(y_test,y_pred)
        report[list(models.keys())[i]]=test_model_score


    return report
```

```
evaluate_model(X_train,X_test,y_train,y_test,models)
```

```
{'random_forest': 0.972972972972973, 'logistic_regression': 0.9864864864864865}
```

```
Start coding or generate with AI.
```