IMPORT SOME LIBRARIES AND READ THE CSV DATASET

```
import pandas as pd
import numpy as np
import scipy as sp
from sklearn import preprocessing



df = pd.read_csv('loan_approval_dataset.csv')
```

```
C:\Users\ashwi\AppData\Local\Temp\ipykernel_21484\679893019.py:1:
DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major
release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type,
and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at
https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

HEAD OF THE DATASET

```
df.head()
```

```
   loan_id   no_of_dependents      education  self_employed
income_annum  \
0        1                  2       Graduate             No
9600000
1        2                  0   Not Graduate            Yes
4100000
2        3                  3       Graduate             No
9100000
3        4                  3       Graduate             No
8200000
4        5                  5   Not Graduate            Yes
9800000

    loan_amount   loan_term   cibil_score
residential_assets_value  \
0      29900000          12           778                    2400000

1      12200000           8           417                    2700000

2      29700000          20           506                    7100000

3      30700000           8           467                   18200000
```

```
4      24200000           20            382                       12400000


    commercial_assets_value   luxury_assets_value
bank_asset_value  \
0                  17600000              22700000              8000000

1                   2200000               8800000              3300000

2                   4500000              33300000             12800000

3                   3300000              23300000              7900000

4                   8200000              29400000              5000000


   loan_status
0     Approved
1     Rejected
2     Rejected
3     Rejected
4     Rejected
```

TAIL OF THE DATASET

```
df.tail()

      loan_id   no_of_dependents      education  self_employed
income_annum  \
4264     4265                  5       Graduate            Yes
1000000
4265     4266                  0   Not Graduate            Yes
3300000
4266     4267                  2   Not Graduate             No
6500000
4267     4268                  1   Not Graduate             No
4100000
4268     4269                  1       Graduate             No
9200000

       loan_amount   loan_term   cibil_score
residential_assets_value  \
4264       2300000          12           317
2800000
4265      11300000          20           559
4200000
4266      23900000          18           457
1200000
4267      12800000           8           780
```

```
8200000
4268      29700000           10            607
17800000

       commercial_assets_value   luxury_assets_value
bank_asset_value  \
4264                     500000                3300000
800000
4265                    2900000               11000000
1900000
4266                   12400000               18100000
7300000
4267                     700000               14100000
5800000
4268                   11800000               35700000
12000000

      loan_status
4264     Rejected
4265     Approved
4266     Rejected
4267     Approved
4268     Approved
```

EXPLORATORY DATA ANAYSIS(EDA)

SHAPE OF THE DATASET

```
print("Rows and Colums of the Dataset : ",df.shape)

Rows and Colums of the Dataset :  (4269, 13)
```

INFO OF THE DATASET

provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   loan_id                   4269 non-null   int64
 1    no_of_dependents         4269 non-null   int64
 2    education                4269 non-null   object
 3    self_employed            4269 non-null   object
 4    income_annum             4269 non-null   int64
```

```
 5    loan_amount               4269 non-null   int64
 6    loan_term                 4269 non-null   int64
 7    cibil_score               4269 non-null   int64
 8    residential_assets_value  4269 non-null   int64
 9    commercial_assets_value   4269 non-null   int64
 10   luxury_assets_value       4269 non-null   int64
 11   bank_asset_value          4269 non-null   int64
 12   loan_status               4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
```

DESCRIPTION IF THE DATASET

The pandas.describe function is used to get a descriptive statistics summary of a given dataframe. This includes mean, count, std deviation, percentiles, and min-max values of all the features.

```
df.describe()
```

```
           loan_id   no_of_dependents   income_annum   loan_amount  \
count  4269.000000        4269.000000   4.269000e+03  4.269000e+03
mean   2135.000000           2.498712   5.059124e+06  1.513345e+07
std    1232.498479           1.695910   2.806840e+06  9.043363e+06
min       1.000000           0.000000   2.000000e+05  3.000000e+05
25%    1068.000000           1.000000   2.700000e+06  7.700000e+06
50%    2135.000000           3.000000   5.100000e+06  1.450000e+07
75%    3202.000000           4.000000   7.500000e+06  2.150000e+07
max    4269.000000           5.000000   9.900000e+06  3.950000e+07

         loan_term   cibil_score   residential_assets_value  \
count  4269.000000   4269.000000               4.269000e+03
mean     10.900445    599.936051               7.472617e+06
std       5.709187    172.430401               6.503637e+06
min       2.000000    300.000000              -1.000000e+05
25%       6.000000    453.000000               2.200000e+06
50%      10.000000    600.000000               5.600000e+06
75%      16.000000    748.000000               1.130000e+07
max      20.000000    900.000000               2.910000e+07

        commercial_assets_value   luxury_assets_value
bank_asset_value
count              4.269000e+03          4.269000e+03
4.269000e+03
mean               4.973155e+06          1.512631e+07
4.976692e+06
std                4.388966e+06          9.103754e+06
3.250185e+06
min                0.000000e+00          3.000000e+05
0.000000e+00
25%                1.300000e+06          7.500000e+06
```

```
2.300000e+06
50%                 3.700000e+06           1.460000e+07
4.600000e+06
75%                 7.600000e+06           2.170000e+07
7.100000e+06
max                 1.940000e+07           3.920000e+07
1.470000e+07
```

df.isnull().sum()

.isnull().sum() returns a DataFrame where each cell is either True or False depending on that cell's null status.

To count the number of nulls in each column we use an aggregate function for summing:

```
df.isnull()

      loan_id   no_of_dependents   education   self_employed
income_annum  \
0       False              False       False           False
False
1       False              False       False           False
False
2       False              False       False           False
False
3       False              False       False           False
False
4       False              False       False           False
False
...       ...                ...         ...             ...
...
4264    False              False       False           False
False
4265    False              False       False           False
False
4266    False              False       False           False
False
4267    False              False       False           False
False
4268    False              False       False           False
False

       loan_amount   loan_term   cibil_score
residential_assets_value  \
0            False       False         False
False
1            False       False         False
False
2            False       False         False
False
```

```
3             False       False         False
False
4             False       False         False
False
...             ...         ...           ...                          ..
.
4264          False       False         False
False
4265          False       False         False
False
4266          False       False         False
False
4267          False       False         False
False
4268          False       False         False
False

       commercial_assets_value   luxury_assets_value
bank_asset_value  \
0                         False                False
False
1                         False                False
False
2                         False                False
False
3                         False                False
False
4                         False                False
False
...                         ...                  ...                  ..
.
4264                      False                False
False
4265                      False                False
False
4266                      False                False
False
4267                      False                False
False
4268                      False                False
False

       loan_status
0            False
1            False
2            False
3            False
4            False
...            ...
```

```
4264         False
4265         False
4266         False
4267         False
4268         False

[4269 rows x 13 columns]
```

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

```
loan_id                     0
 no_of_dependents           0
 education                  0
 self_employed              0
 income_annum               0
 loan_amount                0
 loan_term                  0
 cibil_score                0
 residential_assets_value   0
 commercial_assets_value    0
 luxury_assets_value        0
 bank_asset_value           0
 loan_status                0
dtype: int64
```

ANALYSIS OF EACH COLUMN

if dropna = False it returns the count of null values and if its True then it returns non-null values

```
df.columns
```

```
Index(['loan_id', ' no_of_dependents', ' education', ' self_employed',
       ' income_annum', ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value', ' loan_status'],
      dtype='object')
```

```
df[' education'].value_counts(dropna=False)
```

```
 Graduate        2144
 Not Graduate    2125
Name:  education, dtype: int64
```

```
new_df = df.dropna()
print(new_df)
```

```
      loan_id   no_of_dependents      education  self_employed
income_annum  \
0           1                  2       Graduate             No
9600000
1           2                  0   Not Graduate            Yes
```

```
4100000
2           3                   3       Graduate             No
9100000
3           4                   3       Graduate             No
8200000
4           5                   5   Not Graduate            Yes
9800000
...       ...                 ...            ...            ...
...
4264     4265                   5       Graduate            Yes
1000000
4265     4266                   0   Not Graduate            Yes
3300000
4266     4267                   2   Not Graduate             No
6500000
4267     4268                   1   Not Graduate             No
4100000
4268     4269                   1       Graduate             No
9200000

       loan_amount   loan_term   cibil_score
residential_assets_value  \
0         29900000          12           778
2400000
1         12200000           8           417
2700000
2         29700000          20           506
7100000
3         30700000           8           467
18200000
4         24200000          20           382
12400000
...            ...         ...           ...                          ..
.
4264       2300000          12           317
2800000
4265      11300000          20           559
4200000
4266      23900000          18           457
1200000
4267      12800000           8           780
8200000
4268      29700000          10           607
17800000

       commercial_assets_value   luxury_assets_value
bank_asset_value  \
0                    17600000              22700000
8000000
```

```
1                         2200000                8800000
3300000
2                         4500000               33300000
12800000
3                         3300000               23300000
7900000
4                         8200000               29400000
5000000
...                           ...                    ...                   ..
.
4264                       500000                3300000
800000
4265                      2900000               11000000
1900000
4266                     12400000               18100000
7300000
4267                       700000               14100000
5800000
4268                     11800000               35700000
12000000

      loan_status
0        Approved
1        Rejected
2        Rejected
3        Rejected
4        Rejected
...           ...
4264     Rejected
4265     Approved
4266     Rejected
4267     Approved
4268     Approved

[4269 rows x 13 columns]
```

NORMALIZATION

```
arr = np.array(df[' loan_amount'])
#print(arr)
normalized_arr = preprocessing.normalize([arr])
print(normalized_arr)
```

```
[[0.02595843 0.01059173 0.02578479 ... 0.02074938 0.01111264
0.02578479]]
```

```
from sklearn.preprocessing import MinMaxScaler

normalization_scaling = MinMaxScaler()
```

```
normalized_data = normalization_scaling.fit_transform(df[['
cibil_score']])
print(normalized_data)
```

```
[[0.79666667]
 [0.195     ]
 [0.34333333]
 ...
 [0.26166667]
 [0.8       ]
 [0.51166667]]
```

STANDARDIZATION

```
from sklearn.preprocessing import StandardScaler

scaling = StandardScaler()

standard_scaling = scaling.fit_transform(df[[' loan_amount']])
print(standard_scaling)
```

```
[[ 1.63305171]
 [-0.32441406]
 [ 1.61093345]
 ...
 [ 0.96950399]
 [-0.25805929]
 [ 1.61093345]]
```

Split the data into training, testing and validation sets

```
from sklearn.model_selection import train_test_split
y = df[' loan_status']
X = df.drop(' loan_status',axis = 1)

X_train,X_test,y_train,y_test =
train_test_split(X,y,train_size=0.7,random_state=50)

X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2988 entries, 893 to 1931
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   loan_id                   2988 non-null   int64
 1    no_of_dependents         2988 non-null   int64
 2    education                2988 non-null   object
```

```
 3    self_employed              2988 non-null   object
 4    income_annum               2988 non-null   int64
 5    loan_amount                2988 non-null   int64
 6    loan_term                  2988 non-null   int64
 7    cibil_score                2988 non-null   int64
 8    residential_assets_value   2988 non-null   int64
 9    commercial_assets_value    2988 non-null   int64
 10   luxury_assets_value        2988 non-null   int64
 11   bank_asset_value           2988 non-null   int64
dtypes: int64(10), object(2)
memory usage: 303.5+ KB

y_test.info()

<class 'pandas.core.series.Series'>
Int64Index: 1281 entries, 3149 to 4215
Series name:  loan_status
Non-Null Count  Dtype
--------------  -----
1281 non-null   object
dtypes: object(1)
memory usage: 20.0+ KB
```

Feature Selection

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

X = df.iloc[:,0:20]
y = df.iloc[:,-1]

import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Assuming X is a pandas DataFrame with both numeric and categorical
features
# and 'y' is the target variable

# Separate numeric and categorical features
numeric_features = X.select_dtypes(include=['number']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Apply transformations to the data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', MinMaxScaler(), numeric_features),  # Shift and scale
```

```
numeric features
        ('cat', OneHotEncoder(), categorical_features)
    ])

X_transformed = preprocessor.fit_transform(X)

# Apply SelectKBest to the transformed data
best_features = SelectKBest(score_func=chi2, k=4)
fit = best_features.fit(X_transformed, y)

# Get the selected features
selected_features = fit.transform(X_transformed)

print(selected_features)
```

```
[[0.55555556 0.79666667 1.         0.        ]
 [0.33333333 0.195      0.         1.        ]
 [1.         0.34333333 0.         1.        ]
 ...
 [0.88888889 0.26166667 0.         1.        ]
 [0.33333333 0.8        1.         0.        ]
 [0.44444444 0.51166667 1.         0.        ]]
```

extra

```
from sklearn.preprocessing import StandardScaler

scaling = StandardScaler()

num_col = X_train.select_dtypes(include=['int64','float64']).columns
num_col
X_train[num_col] = scaling.fit_transform(X_train[num_col])

X_train
```

```
       loan_id   no_of_dependents      education  self_employed  \
893  -1.006458           1.455023   Not Graduate            Yes
3749  1.316886          -1.486923   Not Graduate             No
1958 -0.140085           0.866633       Graduate             No
102  -1.649933          -0.310145   Not Graduate            Yes
974  -0.940565          -1.486923       Graduate            Yes
...        ...                ...            ...            ...
3330  0.976031          -0.898534       Graduate             No
70   -1.675965           0.866633   Not Graduate            Yes
132  -1.625528           0.278244   Not Graduate            Yes
2014 -0.094529           0.866633       Graduate            Yes
1931 -0.162049           0.278244       Graduate             No

       income_annum   loan_amount   loan_term   cibil_score  \
893        0.886012      1.061663   -0.506808      0.842310
```

```
3749       1.028895      1.239220    1.235818      1.429435
1958      -0.292771     -0.048062   -0.855333     -1.599198
102       -0.257050     -0.236715    1.584343      0.109858
974       -0.007005      0.373634    0.190242      0.888815
...             ...           ...         ...           ...
3330       0.457364      0.240467    0.190242      0.667917
70        -1.614437     -1.501803    1.584343     -1.506189
132       -1.150068     -1.268760    1.235818     -0.779549
2014       1.350381      2.127000    0.887293      1.423622
1931       0.457364      0.218272    0.887293     -1.227159

      residential_assets_value   commercial_assets_value  \
893                   -0.896438                -0.307055
3749                   1.049592                 2.261685
1958                  -0.170538                 0.225207
102                    0.617141                -0.307055
974                    1.111371                -0.214488
...                         ...                      ...
3330                   0.416360                -0.168204
70                    -1.128108                -1.093876
132                   -1.050884                -1.024450
2014                   1.744603                 1.405438
1931                   0.663475                -0.654182

      luxury_assets_value   bank_asset_value
893               0.188991           1.338644
3749              1.370010           0.415647
1958             -0.362888          -0.599649
102              -0.396000          -0.784248
974              -0.495339           0.477180
...                    ...                ...
3330              1.071996           1.092511
70               -1.444569          -1.338046
132              -1.058254          -0.722715
2014              0.630493           1.769375
1931              0.122765          -0.538116

[2988 rows x 12 columns]
```

```
X_train.isnull().sum()
```

```
loan_id                     0
 no_of_dependents           0
 education                  0
 self_employed              0
 income_annum               0
 loan_amount                0
 loan_term                  0
 cibil_score                0
 residential_assets_value   0
```

```
 commercial_assets_value     0
 luxury_assets_value         0
 bank_asset_value            0
dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

logreg= LogisticRegression()

rfe = RFE(estimator=logreg, n_features_to_select=5) # running RFE with
15 variables as output15

rfe = rfe.fit(X_train,y_train)

col = X_train.columns[rfe.support_]

col
```

```
---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-31-00d091bf4594> in <cell line: 8>()
      6 rfe = RFE(estimator=logreg, n_features_to_select=5) # running
RFE with 15 variables as output15
      7
----> 8 rfe = rfe.fit(X_train,y_train)
      9
     10 col = X_train.columns[rfe.support_]

/usr/local/lib/python3.10/dist-packages/sklearn/feature_selection/
_rfe.py in fit(self, X, y, **fit_params)
    249         """
    250         self._validate_params()
--> 251         return self._fit(X, y, **fit_params)
    252
    253     def _fit(self, X, y, step_score=None, **fit_params):

/usr/local/lib/python3.10/dist-packages/sklearn/feature_selection/
_rfe.py in _fit(self, X, y, step_score, **fit_params)
    258
    259         tags = self._get_tags()
--> 260         X, y = self._validate_data(
    261             X,
    262             y,

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in
_validate_data(self, X, y, reset, validate_separately, **check_params)
    582                 y = check_array(y, input_name="y",
```

```
**check_y_params)
    583             else:
--> 584                 X, y = check_X_y(X, y, **check_params)
    585             out = X, y
    586

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in
check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, multi_output,
ensure_min_samples, ensure_min_features, y_numeric, estimator)
   1104         )
   1105
-> 1106     X = check_array(
   1107         X,
   1108         accept_sparse=accept_sparse,

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples,
ensure_min_features, estimator, input_name)
    877                     array = xp.astype(array, dtype,
copy=False)
    878                 else:
--> 879                     array = _asarray_with_order(array,
order=order, dtype=dtype, xp=xp)
    880             except ComplexWarning as complex_warning:
    881                 raise ValueError(

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py in
_asarray_with_order(array, dtype, order, copy, xp)
    183     if xp.__name__ in {"numpy", "numpy.array_api"}:
    184         # Use NumPy API to support order
--> 185         array = numpy.asarray(array, order=order, dtype=dtype)
    186         return xp.asarray(array, copy=copy)
    187     else:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in
__array__(self, dtype)
   2068
   2069     def __array__(self, dtype: npt.DTypeLike | None = None) ->
np.ndarray:
-> 2070         return np.asarray(self._values, dtype=dtype)
   2071
   2072     def __array_wrap__(

ValueError: could not convert string to float: ' Not Graduate'
```

```
# Assuming you have already fitted the RFE
feature_ranking = rfe.ranking_
selected_features_mask = rfe.support_
```

```
# Print the ranking of features
print("Feature Ranking:")
print(feature_ranking)

# Print the selected features based on the boolean mask
selected_features = X_train.columns[selected_features_mask]
print("\nSelected Features:")
print(selected_features)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-29-a88b6c0017f0> in <cell line: 2>()
      1 # Assuming you have already fitted the RFE
----> 2 feature_ranking = rfe.ranking_
      3 selected_features_mask = rfe.support_
      4
      5 # Print the ranking of features

AttributeError: 'RFE' object has no attribute 'ranking_'
```

```
logreg_ranking = rfe.estimator_.coef_[0]
print("\nLogistic Regression Coefficient Ranking:")
print(logreg_ranking)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-27-7d576da73548> in <cell line: 1>()
----> 1 logreg_ranking = rfe.estimator_.coef_[0]
      2 print("\nLogistic Regression Coefficient Ranking:")
      3 print(logreg_ranking)

AttributeError: 'RFE' object has no attribute 'estimator_'
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

logreg= LogisticRegression()

rfe = RFE(estimator=logreg, n_features_to_select=5) # running RFE with 15 variables as output15

rfe = rfe.fit(X_train,y_train)

col = X_train.columns[rfe.support_]

col
```

```
---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-22-00d091bf4594> in <cell line: 8>()
      6 rfe = RFE(estimator=logreg, n_features_to_select=5) # running
RFE with 15 variables as output15
      7
----> 8 rfe = rfe.fit(X_train,y_train)
      9
     10 col = X_train.columns[rfe.support_]

/usr/local/lib/python3.10/dist-packages/sklearn/feature_selection/
_rfe.py in fit(self, X, y, **fit_params)
    249         """
    250         self._validate_params()
--> 251         return self._fit(X, y, **fit_params)
    252
    253     def _fit(self, X, y, step_score=None, **fit_params):

/usr/local/lib/python3.10/dist-packages/sklearn/feature_selection/
_rfe.py in _fit(self, X, y, step_score, **fit_params)
    258
    259         tags = self._get_tags()
--> 260         X, y = self._validate_data(
    261             X,
    262             y,

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in
_validate_data(self, X, y, reset, validate_separately, **check_params)
    582                 y = check_array(y, input_name="y",
**check_y_params)
    583             else:
--> 584                 X, y = check_X_y(X, y, **check_params)
    585             out = X, y
    586

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in
check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, multi_output,
ensure_min_samples, ensure_min_features, y_numeric, estimator)
   1104         )
   1105
-> 1106     X = check_array(
   1107         X,
   1108         accept_sparse=accept_sparse,

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in
check_array(array, accept_sparse, accept_large_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples,
```

```
ensure_min_features, estimator, input_name)
    877                     array = xp.astype(array, dtype,
copy=False)
    878                 else:
--> 879                     array = _asarray_with_order(array,
order=order, dtype=dtype, xp=xp)
    880             except ComplexWarning as complex_warning:
    881                 raise ValueError(

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py in
_asarray_with_order(array, dtype, order, copy, xp)
    183     if xp.__name__ in {"numpy", "numpy.array_api"}:
    184         # Use NumPy API to support order
--> 185         array = numpy.asarray(array, order=order, dtype=dtype)
    186         return xp.asarray(array, copy=copy)
    187     else:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in
__array__(self, dtype)
   2068
   2069     def __array__(self, dtype: npt.DTypeLike | None = None) ->
np.ndarray:
-> 2070         return np.asarray(self._values, dtype=dtype)
   2071
   2072     def __array_wrap__(

ValueError: could not convert string to float: ' Not Graduate'
```