```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from IPython.display import display, Image

warnings.filterwarnings("ignore")

pd.set_option('max.columns',100)
pd.set_option('max.rows',500)
```

```
[4] data = pd.read_csv('Titanic.csv')
    display(data.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
def find_missing_percent(data , showresult = True):
    miss_df = pd.DataFrame({'ColumnName':[],'TotalMissingVals':[],'PercentMissing':[]})
    for col in data.columns:
        sum_miss_val = data[col].isnull().sum()
        percent_miss_val = round((sum_miss_val/data.shape[0])*100,2)
        missinginfo = {"ColumnName" : col, "TotalMissingVals" : sum_miss_val, "PercentMissing" : percent_miss_val}
        miss_df = miss_df.append(missinginfo, ignore_index = True)

    miss_df = miss_df[miss_df["PercentMissing"] > 0.0]
    miss_df = miss_df.reset_index(drop = True)
    miss_features = miss_df["ColumnName"].values
    if(showresult):
        print(data.shape)
        display(data.head())
        display(miss_df)
    return miss_df
```

```
[26] miss_df = find_missing_percent(data)
```

```
(891, 12)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

	ColumnName	TotalMissingVals	PercentMissing
0	Age	177.0	19.87
1	Cabin	687.0	77.10
2	Embarked	2.0	0.22

```
[27] def listwise_deletion(data):
        for col in data.columns:
            miss_ind = data[col][data[col].isnull()].index
            data = data.drop(miss_ind, axis = 0)
        return data
```

```
[28] data_lwd = listwise_deletion(data)
     miss_df = find_missing_percent(data_lwd)
```

```
(183, 12)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S

	ColumnName	TotalMissingVals	PercentMissing

```
[29] numeric_cols = data.select_dtypes(['float','int']).columns
     categoric_cols = data.select_dtypes('object').columns
     print(f"Numeric Columns : {numeric_cols}")
     print(f"Categoric Columns : {categoric_cols}")
```

```
Numeric Columns : Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare'], dtype='object')
Categoric Columns : Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

```
[30] def mean_imputation(data_numeric):
         for col in data_numeric.columns:
             mean = data_numeric[col].mean()
             data_numeric[col] = data_numeric[col].fillna(mean)
         return data_numeric
     def mode_imputation(data_categoric):
         for col in data_categoric.columns:
             mode = data_categoric[col].mode().iloc[0]
             data_categoric[col] = data_categoric[col].fillna(mode)
         return data_categoric
```

```
[31] data_numeric = data[numeric_cols]
     data_numeric_mean_imp = mean_imputation(data_numeric)
     data_categoric = data[categoric_cols]
     data_categoric_mode_imp = mode_imputation(data_categoric)

     data_imputed_value = pd.concat([data_numeric_mean_imp, data_categoric_mode_imp], axis = 1)
     miss_df = find_missing_percent(data_imputed_value)
```

```
(891, 12)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Name	Sex	Ticket	Cabin	Embarked
0	1	0	3	22.0	1	0	7.2500	Braund, Mr. Owen Harris	male	A/5 21171	B96 B98	S
1	2	1	1	38.0	1	0	71.2833	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	PC 17599	C85	C
2	3	1	3	26.0	0	0	7.9250	Heikkinen, Miss. Laina	female	STON/O2. 3101282	B96 B98	S
3	4	1	1	35.0	1	0	53.1000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	C123	S
4	5	0	3	35.0	0	0	8.0500	Allen, Mr. William Henry	male	373450	B96 B98	S

	ColumnName	TotalMissingVals	PercentMissing

```
[32] import xgboost
     from sklearn.experimental import enable_iterative_imputer
     from sklearn.impute import IterativeImputer
     from sklearn.preprocessing import OrdinalEncoder
     from sklearn.ensemble import (GradientBoostingRegressor, GradientBoostingClassifier)
```

```
[33] def find_missing_index(data_numeric_xgboost, target_cols):
         miss_index_dict = {}
         for tcol in target_cols:
             index = data_numeric_xgboost[tcol][data_numeric_xgboost[tcol].isnull()].index
             miss_index_dict[tcol] = index
         return miss_index_dict

     def xgboost_imputation(data_numeric_xgboost, target_cols, miss_index_dict):
         predictors = data_numeric_xgboost.drop(target_cols, axis =1)
         for tcol in target_cols:
             y = data_numeric_xgboost[tcol]
             y = y.fillna(y.mean())
             xgb = xgboost.XGBRegressor(objective="reg:squarederror", random_state=42)
             xgb.fit(predictors, y)
             predictions = pd.Series(xgb.predict(predictors),index= y.index)
             index = miss_index_dict[tcol]
             data_numeric_xgboost[tcol].loc[index] = predictions.loc[index]
         return data_numeric_xgboost
```

```
[34] miss_df = find_missing_percent(data, showresult = False)
     miss_features = miss_df["ColumnName"].values
     target_cols = [feature for feature in miss_features if feature in numeric_cols]
     print(target_cols)

     data_numeric_xgboost = data[numeric_cols]
     miss_index_dict = find_missing_index(data_numeric_xgboost, target_cols)
     data_numeric_xgboost = xgboost_imputation(data_numeric_xgboost, target_cols, miss_index_dict)
     data_imputed_xgboost = pd.concat([data_numeric_xgboost, data_categoric_mode_imp], axis = 1)

     miss_df = find_missing_percent(data_imputed_xgboost)
```

```
['Age']
(891, 12)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Name	Sex	Ticket	Cabin	Embarked
0	1	0	3	22.0	1	0	7.2500	Braund, Mr. Owen Harris	male	A/5 21171	B96 B98	S
1	2	1	1	38.0	1	0	71.2833	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	PC 17599	C85	C
2	3	1	3	26.0	0	0	7.9250	Heikkinen, Miss. Laina	female	STON/O2. 3101282	B96 B98	S
3	4	1	1	35.0	1	0	53.1000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	C123	S
4	5	0	3	35.0	0	0	8.0500	Allen, Mr. William Henry	male	373450	B96 B98	S

	ColumnName	TotalMissingVals	PercentMissing

```
[35] def mice_imputation_numeric(train_numeric):
        iter_imp_numeric = IterativeImputer(GradientBoostingRegressor())
        imputed_train = iter_imp_numeric.fit_transform(train_numeric)
        train_numeric_imp = pd.DataFrame(imputed_train, columns = train_numeric.columns, index= train_numeric.index)
        return train_numeric_imp

    def mice_imputation_categoric(train_categoric):
        ordinal_dict={}
        for col in train_categoric:
            ordinal_dict[col] = OrdinalEncoder()
            nn_vals = np.array(train_categoric[col][train_categoric[col].notnull()]).reshape(-1,1)
            nn_vals_arr = np.array(ordinal_dict[col].fit_transform(nn_vals)).reshape(-1,)
            train_categoric[col].loc[train_categoric[col].notnull()] = nn_vals_arr

        iter_imp_categoric = IterativeImputer(GradientBoostingClassifier(), max_iter =5, initial_strategy='most_frequent')
        imputed_train = iter_imp_categoric.fit_transform(train_categoric)
        train_categoric_imp = pd.DataFrame(imputed_train, columns =train_categoric.columns,index = train_categoric.index).astype(int)

        for col in train_categoric_imp.columns:
            oe = ordinal_dict[col]
            train_arr= np.array(train_categoric_imp[col]).reshape(-1,1)
            train_categoric_imp[col] = oe.inverse_transform(train_arr)

        return train_categoric_imp
```

```
data_numeric_imp  = mice_imputation_numeric(data_numeric)
data_categoric_imp = mice_imputation_categoric(data_categoric)

data_imputed_mice = pd.concat([data_numeric_imp, data_categoric_imp], axis = 1)
miss_df = find_missing_percent(data_imputed_mice)
```

```
(891, 12)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Name	Sex	Ticket	Cabin	Embarked
0	1.0	0.0	3.0	22.0	1.0	0.0	7.2500	108.0	1.0	523.0	47.0	2.0
1	2.0	1.0	1.0	38.0	1.0	0.0	71.2833	190.0	0.0	596.0	81.0	0.0
2	3.0	1.0	3.0	26.0	0.0	0.0	7.9250	353.0	0.0	669.0	47.0	2.0
3	4.0	1.0	1.0	35.0	1.0	0.0	53.1000	272.0	0.0	49.0	55.0	2.0
4	5.0	0.0	3.0	35.0	0.0	0.0	8.0500	15.0	1.0	472.0	47.0	2.0

ColumnName	TotalMissingVals	PercentMissing

```
[44] skew_limit = 0.5
     skew_vals = data_modelling[numeric_cols].skew()
     skew_cols = (skew_vals
                  .sort_values(ascending=False)
                  .to_frame()
                  .rename(columns={0:'Skew'})
                  .query('abs(Skew) > {0}'.format(skew_limit)))
     display(skew_cols.T)
```

	Fare	Pclass	Parch	SibSp	Survived
Skew	2.715883	2.681459	1.519305	1.443019	-0.739427

```python
def NormalizeSkewedFeatures(data_modelling):
    from scipy.special import boxcox1p
    from scipy.stats import boxcox_normmax
    for col in skew_cols.index:
        if(col != 'Fare'):
            try:
                data_modelling[col] = boxcox1p(data_modelling[col], boxcox_normmax(data_modelling[col] + 1))
            except:
                print(f"column {col} can not apply BoxCox")
                continue
    return data_modelling

data_modelling = NormalizeSkewedFeatures(data_modelling)
data_modelling["Fare"] = np.log1p(data_modelling["Fare"])
```

```python
def FeatureEncoding(data_modelling):
    data_modelling = pd.get_dummies(data_modelling, columns=categoric_cols, drop_first=True)
    return data_modelling

data_modelling = FeatureEncoding(data_modelling)
display(data_modelling.head())
print(data_modelling.shape)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Name_Allison, Master. Hudson Trevor	Name_Allison, Miss. Helen Loraine	Name_Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	Name_Anderson, Mr. Harry	Name_Andrews, Miss. Kornelia Theodosia	Name_Andrews, Mr. Thomas Jr	N
1	2	7.38668	0.440686	38.0	0.741533	0.000000	4.280593	0	0	0	0	0	0	
3	4	7.38668	0.440686	35.0	0.741533	0.000000	3.990834	0	0	0	0	0	0	
6	7	0.00000	0.440686	54.0	0.000000	0.000000	3.967694	0	0	0	0	0	0	
10	11	7.38668	0.605054	4.0	0.741533	0.731053	2.873565	0	0	0	0	0	0	
11	12	7.38668	0.440686	58.0	0.000000	0.000000	3.316003	0	0	0	0	0	0	

5 rows × 450 columns

(183, 450)

```python
[57] from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import mean_squared_error, r2_score

     def DataSplitTrainTest(data_modelling):
         train = data_modelling.copy()
         X = train.drop('Fare', axis=1)
         y = train['Fare']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=12345)
         print("Train Data", X_train.shape)
         print("Test Data", X_test.shape)
         return X_train, X_test, y_train, y_test

     X_train, X_test, y_train, y_test = DataSplitTrainTest(data_modelling)
```

```
Train Data (128, 449)
Test Data (55, 449)
```

```
[58] def BuildLassoModel(X_train, X_test, y_train, y_test):
         lasso = Lasso(max_iter = 100000, normalize = True)
         lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
         lassocv.fit(X_train, y_train)

         lasso.set_params(alpha=lassocv.alpha_)
         lasso.fit(X_train, y_train)

         print('The Lasso:')
         print("Alpha =", lassocv.alpha_)
         print("RMSE =", mean_squared_error(y_test, lasso.predict(X_test), squared=False))
         print("R2 Score = ", r2_score(y_test, lasso.predict(X_test)))
         return lasso

     lasso = BuildLassoModel(X_train, X_test, y_train, y_test)
```

```
The Lasso:
Alpha = 0.00033599657046442583
RMSE = 0.5124260530064272
R2 Score =  0.6342195491576293
```

```
def BuildRidgeModel(X_train, X_test, y_train, y_test):
    alphas = np.geomspace(1e-9, 5, num=100)
    ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
    ridgecv.fit(X_train, y_train)

    ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
    ridge.fit(X_train, y_train)

    print('Ridge Regression:')
    print("Alpha =", ridgecv.alpha_)
    print("RMSE =", mean_squared_error(y_test, ridge.predict(X_test), squared=False))
    print("R2 Score = ", r2_score(y_test, lasso.predict(X_test)))
    return ridge

ridge = BuildRidgeModel(X_train, X_test, y_train, y_test)
```

```
Ridge Regression:
Alpha = 1e-09
RMSE = 0.4761028335384219
R2 Score =  0.6342195491576293
```