IMPORTING DATASET:

```
import os
import pandas as pd
#load data
data0=pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')
```

IDENTIFYING DATA TYPE OF EACH COLUMN:

```
total_data = pd.concat([data0], ignore_index=True)
type(total_data)
total_data['X'].dtype
```

HANDLING MISSING DATA:

```
total_data = pd.concat([data_csv], ignore_index=True)
numerical_data=total_data.select_dtypes(exclude=[object])
print(numerical_data.shape)
```

DATATYPES HANDLING:

```
data_frame.month.replace(('jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec'),(1
,2,3,4,5,6,7,8,9,10,11,12), inplace=True)
data_csv.day.replace(('mon','tue','wed','thu','fri','sat','sun'),(1,2,3,4,5,6,7), inplace=True)

print("Head:", data_frame.head())

print("Statistical Description:", data_frame.describe())

print("Shape:", data_frame.shape)

print("Data Types:", data_frame.dtypes)


data_frame.isna().any()

RH = data_csv.month
```

```
np.unique(RH)

print(RH)

RH = set(RH)
print(RH)

print(data_frame_csv)
```

LIST:

```
month[3] = 'harsini'
month
```

SEARCHING LIST:

LIST FUNCTIONS:

```
len(month)

sorted(month)
```

Tuple:

```
day = ('mon','tue','wed','thu','fri','sat','sun')
```

SET:

```
set = {'jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov'}
type(set)
```

Add and remove items from a set with add() and remove() respectively:

```
set.add("dec")
set
```

Python sets support many common mathematical set operations like union, intersection, difference.

```
set1 = {'jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec'}
```

```
set2 = {'mon','tue','wed','thu','fri','sat','sun'}

set1.union(set2)
```

DICTIONARY:

Create a dictionary with a comma-separated list of key: value pairs within curly braces:

```
dict = {"month": "mar",
        "day": "fri"}
print(dict)
```

Add new items to an existing dictionary with the following syntax:

```
dict["RH"] = "97"
print(dict)
```

Delete existing key: value pairs with del:

```
del dict["RH"]

print(dict)
```

DICTIONARY FUNCTIONS:

```
dict.values()
```

ANALYSING CATEGORICAL COLUMNS:

```
dfa = data_csv.drop(columns='area')
cat_columns = dfa.select_dtypes(include='object').columns.tolist()
num_columns = dfa.select_dtypes(exclude='object').columns.tolist()
```

BAR CHART:

```
plt.figure(figsize=(16,10))
for i,col in enumerate(cat_columns,1):
```

```
    plt.subplot(2,2,i)
    sns.countplot(data=dfa,y=col)  #countplot:count of each month/day in month/day
columns
    plt.subplot(2,2,i+2)
    data_csv[col].value_counts().plot.bar() #freq of each month/day in month/day columns
    plt.ylabel(col)
    plt.xlabel('% distribution per category')
plt.show()
```

Quantile plot for temp and wind:

```
x=data_csv['temp']

y=data_csv['wind']
stats.probplot(x,dist="norm", plot=pylab)
pylab.show()
stats.probplot(y, dist="norm", plot=pylab)
pylab.show()
```

HISTOGRAM:

```
plt.hist2d(x,y)
```

SCATTER PLOT:

```
data_csv.plot(kind='scatter', x='X', y='Y', alpha=0.2,
s=20*data_csv['area'],figsize=(10,6))
plt.xlabel('X cordinates of regions',color='red',fontsize=15)
plt.ylabel('Y cordinates of regions',color='red',fontsize=15)
plt.title('Burnt area in different regions',color='blue',fontsize=18)

sns.jointplot(x,y)
```

BAR WITH XTICKS:

```
def area_cat(area):      # grouping damage category based on amount of area burned.
    if area == 0.0:
        return "No damage"
```

```
    elif area <= 1:
        return "low"
    elif area <= 25:
        return "moderate"
    elif area <= 100:
        return "high"
    else:
        return "very high"
data_csv['damage_category'] = data_csv['area'].apply(area_cat)
for col in cat_columns:

cross=pd.crosstab(index=data_csv['damage_category'],columns=data_csv[col],normalize
='index')
    cross.plot.barh(stacked=True,rot=40,cmap='plasma')
    plt.xlabel('% distribution per category')
    plt.xticks(np.arange(0,1.1,0.1))
    plt.title("Forest Fire damage each {}".format(col))
plt.show()
```

PIE CHART:

```
areaburnt=data_csv[data_csv['area']>0]areaburnt
areaburnt.groupby('month')['area'].agg('count').plot(kind='pie',title='Monthly analysis of
burnt area',figsize=(9,9),explode=[0,0.1,0,0,0,0,0,0,0,0.1],autopct='%0.1f%%')
plt.show()
```

REG PLOT:

```
import seaborn as sns
sns.set(style="darkgrid")
sns.regplot(x=total_data['rain'],y=total_data['temp'])
sns.regplot(x=total_data['rain'],y=total_data['temp'],fit_reg=False)

sns.regplot(x=total_data['rain'],y=total_data['temp'], fit_reg=False, marker="*")

sns.lmplot(x='RH',y='wind',data=total_data, fit_reg=False, hue='X',legend=True,
palette='Set1')
```

```
sns.distplot(total_data['RH'])
sns.distplot(total_data['RH'],kde=False)
sns.distplot(total_data['RH'],kde=False,bins=5)

sns.countplot(x="X",data=total_data)

sns.boxplot(y=total_data["RH"])

sns.boxplot(x=total_data["RH"],y=total_data["wind"])

sns.pairplot(total_data, kind="scatter", hue="X")
plt.show()

 import os
 import pandas as pd
 from sklearn.preprocessing import StandardScaler
 from sklearn.decomposition import PCA
 import numpy as np
 import matplotlib.pyplot as plt
 import seaborn as sns
 from sklearn.preprocessing import MinMaxScaler
 data_csv=pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')

 import pandas as pd
 import numpy as np
 import seaborn as sns
 import matplotlib.pyplot as plt
 from sklearn.feature_selection import RFE
 from sklearn.ensemble import ExtraTreesRegressor
 from sklearn.linear_model import LinearRegression
 from sklearn.linear_model import Lasso
 from sklearn.linear_model import Ridge
 from sklearn.metrics import explained_variance_score
 from sklearn.metrics import mean_absolute_error
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
data_csv=pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')

print("correlation:",data_csv.corr(method='pearson'))
```

DESCRIBING THE DATASET:

```
data_csv.describe()
data.describe(include="O")
var=data.corr()
var
```

CORRELATION:

```
import seaborn as sns
correlation=data_csv.corr()
sns.heatmap(correlation)
```

LINEAR CORRELATION:

```
plt.matshow(data_csv.corr())
plt.colorbar()
plt.show()
```

Spearman rank correlation is more robust to the effect of outliers than Pearson's correlations coefficient.

SPEARMAN METHOD OF CORRELATION:

```
corr_matrix = data_csv.corr(method='spearman')
corr_matrix

ax = plt.figure(figsize=(12,8))
ax = sns.heatmap(corr_matrix, cmap='PiYG')
```

#correlation with area

```
corr_matrix.area.sort_values(ascending=False)
```

Principal Component Analysis(PCA):

```
data_csv = pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')
from sklearn.preprocessing import StandardScaler
features=['X', 'Y']
x=data_csv.loc[:,features].values

y=data_csv.loc[:,['RH']].values

x=StandardScaler().fit_transform(x)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)

import matplotlib.pyplot as plt
fig = plt.figure(figsize = (8,8))

scaler=StandardScaler()
scaler.fit(x)
StandardScaler()
scaled_data=scaler.transform(x)
scaled_data

data_csv.dropna()
print(data.shape)
print(data.columns)
data_csv.groupby('RH').mean()

data_csv['RH']=data_csv['RH'].astype('category')
data_csv['RH']=data_csv['RH'].cat.codes
data_csv
```

READING OF DATASET:

```
data_csv=pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')

data_csv.describe()
```

UNDERSTANDING CONFIDENCE INTERVAL :

```
import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KernelDensity

def kde_sklearn(x, x_grid, bandwidth=0.2, **kwargs):
    kde_skl = KernelDensity(bandwidth=bandwidth, **kwargs)
    kde_skl.fit(x[:, np.newaxis])
    log_pdf = kde_skl.score_samples(x_grid[:, np.newaxis])
    return np.exp(log_pdf)

data_csv=pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')
df = data_csv['wind'].copy()
plt.hist(df, bins=100,alpha=0.9, density=True)
x_grid = np.linspace(-1, 1, 100)
pdf_engagement = kde_sklearn(df, x_grid, bandwidth=0.007)

plt.hist(df, bins=100,alpha=0.9, density=True)
x_grid = np.linspace(-1, 1, 100)
pdf_engagement = kde_sklearn(df, x_grid, bandwidth=0.007)
plt.plot(x_grid, pdf_engagement,alpha=0.9, lw=5, color='r')
plt.xlabel("wind")
plt.ylabel("temp")
plt.show()

mean = np.mean(df)
std = np.std(df)
print("""
Population mean: %.5f
```

```
Population std: %.5f
Population size: %i
"""%(mean, std, len(df)))
```

EXAMINE SAMPLING DISTRIBUTION:

```
sample_size = 300
n_trials = 50000
samples = np.array([np.random.choice(df, sample_size)
for _ in range(n_trials)])
means = samples.mean(axis=1)
sample_mean = np.mean(means)
sample_std = np.std(means)
analytical_std = std / np.sqrt(sample_size)
print("""
sampling distribution mean: %.5f
sampling distribution std: %.5f
analytical std: %.5f
"""%(sample_mean, sample_std, analytical_std))
```

NORMAL SAMPLING DISTRIBUTION:

```
from scipy.stats import t
z = 1.96
plt.hist(means, bins=50, alpha=0.9, density=True)
plt.axvline(sample_mean - 1.96 * sample_std, color='r')
plt.axvline(sample_mean + 1.96 * sample_std, color='r')
plt.xlabel('sample_mean')
plt.ylabel('wind')
plt.show()
```

SKEWED SAMPLING DISTRIBUTION:

```
print("lower tail: %.2f%%"%(100 * sum(means < sample_mean - 1.96 * sample_std) /
len(means)))
print("upper tail: %.2f%%"%(100 * sum(means > sample_mean + 1.96 * sample_std) /
len(means)))

import pylab
```

```
import scipy.stats as stats
stats.probplot(means, dist="norm", plot=pylab)
pylab.show()
```

CONFIDENCE INTERVAL:

```
z = 1.96
se = samples.std(axis=1) / np.sqrt(sample_size)
ups = means + z * se
los = means - z * se
success = np.mean((mean >= los) & (mean <= ups))
fpr = np.mean((mean < los) | (mean > ups))
print("False positive rate: %.3f"%fpr)
False positive rate: 0.052

n_points = 10000
# plt.figure(figsize=(14, 6))
plt.scatter(list(range(len(ups[:n_points]))), ups[:n_points], alpha=0.9)
plt.scatter(list(range(len(los[:n_points]))), los[:n_points], alpha=0.9)
plt.axhline(y=0.0551)
plt.xlabel("wind")
plt.ylabel("sample_mean")

n_points = 10000
# plt.figure(figsize=(14, 6))
plt.scatter(list(range(len(ups[:n_points]))), ups[:n_points], alpha=0.9)
plt.scatter(list(range(len(los[:n_points]))), los[:n_points], alpha=0.9)
plt.axhline(y=0.0551)
plt.xlabel("sample")
plt.ylabel("sample_mean")
```

Testing :

1.Normality Tests :

Anderson-Darling Normality Tests :

```
from scipy.stats import anderson
data_csv=pd.read_csv('C:\\18C041\\Data Science Using Python\\forestfires.csv')
data =data_csv['rain']
```

```
result = anderson(data)
print('stat=%.3f' % (result.statistic))
for i in range(len(result.critical_values)):
 sl, cv = result.significance_level[i], result.critical_values[i]
 if result.statistic < cv:
  print('Probably Gaussian at the %.1f%% level' % (sl))
else:
  print('Probably not Gaussian at the %.1f%% level' % (sl))
```

Correlation Tests :

2.Correlation Tests: Chi-Squared Test - Tests whether two categorical variables are related or independent.

```
from scipy.stats import chi2_contingency
table = [data_csv['temp'], data_csv['DC']]
stat, p, dof, expected = chi2_contingency(table)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably independent')
else:
 print('Probably dependent')
```

3.Stationary Tests: Augmented Dickey-Fuller Unit Root Test - Tests whether a time series has a unit root.

```
from statsmodels.tsa.stattools import adfuller
data = data_csv['FFMC']
stat, p, lags, obs, crit, t = adfuller(data)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably not Stationary')
else:
 print('Probably Stationary')
```

4. Parametric Statistical Hypothesis Tests One Way - Analysis of Variance Test (ANOVA)

```
from scipy.stats import f_oneway
data = data_csv['temp'].copy();
```

```
data1 = data_csv['rain'].copy();
stat, p = f_oneway(data,data1)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

5. Nonparametric Statistical Hypothesis Tests: Kruskal-Wallis H Test - Tests whether the distributions of two or more independent samples are equal or not.

```
from scipy.stats import kruskal
data1 = data_csv['HNR'].copy();
data2 = data_csv['RPDE'].copy();
stat, p = kruskal(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

FIRST ANOVA TEST

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import random
import statsmodels.api as sm
import statsmodels.stats.multicomp
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
data_csv['FFMC'].sort_values()

data_csv.head()
```

```python
df_anova=data_csv[['DC','rain']]
rp=pd.unique(data_csv.rain.values)
a={rps:data_csv['DC'][df_anova.rain==rps]for rps in rp}
print(a)

F,p=stats.f_oneway(data_csv['FFMC'],data_csv['temp'],data_csv['area'])
print(p)

if p<0.05:
    print("we reject null hypothesis")
else:
    print("we accept null hypothesis")


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder


df = pd.read_csv('C:\\18C037\\data science using python\\forestfires.csv')
# By this we are able to see the features in the dataset and sample data.
df.head()

#we can see that there are many features of type object.
df.dtypes

# Finding null values
```

```
df.isna().sum()

#we can see no null values in the dataset
df=df.fillna(0)
df.isna().sum()

#In this dataset there are so many categorical features. We  have to change the function
and perform logistic regression.

le = LabelEncoder()

le.fit(df.month.drop_duplicates())
df.month = le.transform(df.month)

le.fit(df.day.drop_duplicates())
df.day = le.transform(df.day)

le.fit(df.FFMC.drop_duplicates())
df.FFMC = le.transform(df.FFMC)

le.fit(df.DMC.drop_duplicates())
df.DMC = le.transform(df.DMC)

le.fit(df.DC.drop_duplicates())
df.DC = le.transform(df.DC)

le.fit(df.ISI.drop_duplicates())
df.ISI = le.transform(df.ISI)

le.fit(df.temp.drop_duplicates())
df.temp = le.transform(df.temp)
```

```
le.fit(df.RH.drop_duplicates())
df.RH = le.transform(df.RH)

le.fit(df.wind.drop_duplicates())
df.wind = le.transform(df.wind)

le.fit(df.rain.drop_duplicates())
df.rain = le.transform(df.rain)

le.fit(df.area.drop_duplicates())
df.area = le.transform(df.area)

#Now we can change the categorical features into  numerical
df.head()

x=df.drop(['area','RH','rain'], axis=1)

y=df['area']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
x_train

y_train
```

PRINCIPAL COMPONENT ANALYSIS(PCA):

```
from sklearn.decomposition import PCA

pca=PCA(n_components=2)

pca.fit(x_train)
```

```
x_pca=pca.transform(x_train)

x_train.shape

x_pca.shape

plt.figure(figsize=(5,4))
plt.scatter(x_pca[:,0],x_pca[:,1],c=y_train)
plt.xlabel('PCA1')
plt.ylabel('PCA2')

logreg = LogisticRegression(solver='liblinear', random_state=0)

logreg.fit(x_pca, y_train)

x_te=pca.fit_transform(x_test)
y_pred_test=logreg.predict(x_te)
print('MODEL ACCURACY SCORE:{0:0.4f}'.format(accuracy_score(y_test,
y_pred_test)))
```

Dimensionality reduction with SVD:

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=2)

x_sv=svd.fit_transform(x_train)
logreg = LogisticRegression(solver='liblinear', random_state=0)

logreg.fit(x_sv, y_train)

x_te=svd.fit_transform(x_test)
```

```
y_pred_test=logreg.predict(x_te)
print('MODEL ACCURACY SCORE:{0:0.4f}'.format(accuracy_score(y_test,
y_pred_test)))
```