```
#Importing the library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from scipy.stats import norm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression # use f_regression for regr
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
%matplotlib inline
```

```
#Load the dataset
df=pd.read_csv('dataset/Lab02_prepared.csv')
```

```
#peek of dataset
df.head()
```

```
#about the dataset
df.info()
```

```
#checking for the null values
df.isnull().sum()
```

b. Correlation & heatmap

```
#correlation of the features
df.corr()
```

```
#ploting the heatmap
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),
            cbar=True,
            annot=True,
            square=True,
            fmt='.1f',
            annot_kws={'size':12},
            yticklabels=df.columns,
            xticklabels=df.columns)
```

- Observations: This shows that 'log_inst_review' has high correlation with 'log_enrollment' and 'log_number_ratings'. 'l
correlation with 'log_enrollment' and 'log_number_ratings'. 'log_inst_student' and 'log_inst_student' has 1.0 correlatio
have same data.

```
#Multicollinearity analysis
df_multi=df
df_independent = df_multi.select_dtypes(include=np.number).drop('log_inst_review', axis=1)

# create a dataframe for VIF data
vif_data = pd.DataFrame()
vif_data["feature"] = df_independent.columns

vif_data["vif"] = [variance_inflation_factor(df_independent.values, i)
                   for i in range(len(df_independent.columns))
]

print(vif_data)
```

```
scaled_features = StandardScaler().fit_transform(df_independent)
scaled_df1 = pd.DataFrame(scaled_features, columns=df_independent.columns)

# create a dataframe for VIF data
vif_data = pd.DataFrame()
vif_data["feature"] = scaled_df1.columns

vif_data["vif"] = [variance_inflation_factor(scaled_df1.values, i)
                   for i in range(len(scaled_df1.columns))
]

print(vif_data)
```

Observations: It shows the high multicollinearity in the Log_enrollment and log_number_ratings. we should have
or analysis .

```
scaled_df1.drop('log_enrollment', axis=1, inplace=True)
# create a dataframe for VIF data
vif_data = pd.DataFrame()
vif_data["feature"] = scaled_df1.columns

vif_data["vif"] = [variance_inflation_factor(scaled_df1.values, i)
                   for i in range(len(scaled_df1.columns))
]

print(vif_data)
```

After dropping the log_enrollment column the data set as no multicollinearity , we should have dropped it from t
mentioned in the steps. I wouldn't drop it .

c. Univariate Analysis

```
#Univariate analysis
 #Distribution plot
sns.displot(df.log_inst_student)
```

- Observation: this graph is left skewed. it has a long left tail.

```
#find the index and drop
x=df[df.log_inst_student<7].index
print(x)
df.drop(x,inplace=True)
# ploting the graph
sns.distplot(df.log_inst_student,fit=norm)
```

d. Multivariate Analysis

```
#MultiVariate Analysis
sns.scatterplot(x='log_inst_student',y='log_inst_review',data=df)
```

- Observation: They are growing in the same direction and not disperse. Means they are

```
sns.scatterplot(x='log_enrollment',y='log_inst_review',data=df)
```

- Observation: They are growing in same direction but they are alot disperse.

4.Feature Selection

a. Created column of the dataframe to a new variable called target

```
#create a copy of data frame
df_copy=df
#target
target=df_copy.log_inst_review
features=df_copy.drop('log_inst_review',axis=1)
```

```
df_copy.info()
```

b. Correlation Based Selection(Threshold)

```
#imported the function
%run functions/Functions_KM146384.ipynb
```

```
#using threshold value to select the best values
#called the method and assigned it to df
df_correlation=select_threshold(df_copy)
df_correlation.info()
```

c. Variance Threshold Selection

```
df_copy.info()
```

```
#ran the function
%run functions/Functions_KM146384.ipynb
```

```
#calling the function and using the variance threshold
df_variance=selected_variance(features,0.2)
```

```
#printing the df
df_variance.info()
```

d. Select K-Best method

```
#ran the function
%run functions/Functions_KM146384.ipynb
```

```
#calling the function
df_selkbest=select_best(features,9)
```

```
#display the dataset
df_selkbest.head()
```

```
df_selkbest.info()
```

```
for method in ['correlation', 'variance', 'selkbest']:

    if method=='correlation':
        features=df_correlation
    elif method=='variance':
        features=df_variance
    elif method=='selkbest':
        features=df_selkbest

    for transforms in ['No Transformation', 'Polynomial Transformation']:
                if(transforms=='Polynomial Transformation'):
                    %run functions/Functions_KM146384.ipynb
                    transformed_df=make_poly(features)
                elif(transforms=='No Transformation'):
                        transformed_df=features
    X_train, X_test, Y_train, Y_test = train_test_split(transformed_df,target, test_size=0.25
                                                        random_state=42)

    # Create a scaler instance
    scaler = StandardScaler()
            # Fit the scaler on the training data
    scaled_train_X=scaler.fit_transform(X_train)
    scaled_test_X =scaler.fit_transform(X_test)

            #remake the dataframe
    X_train_scaled=pd.DataFrame(scaled_train_X,columns=X_train.columns)
    X_test_scaled=pd.DataFrame(scaled_train_X,columns=X_test.columns)

    # build the model
    model = LinearRegression()
    model.fit(X_train, Y_train)
    # get the predictions for the test dataset
    Y_pred = model.predict(X_test)
    # R-squared
    r2 = model.score(X_test, Y_test)
    print("R2 score is ", r2)
    from sklearn.metrics import mean_squared_error
    # RMSE
    rmse = mean_squared_error(Y_test, Y_pred, squared=False)
    print("RMSE is ", rmse)
    #appending the values to lists
    fselection_list.append(method)
    ftransform_list.append(transforms)
    r2_list.append(r2)
    rmse_list.append(rmse)
```

6. Linear Regression Model with Ridge

```
target=df_copy.log_inst_review
features=df_copy.drop('log_inst_review',axis=1)
```

```
# split the data into 75:25 ration for train and test
X_train, X_test, Y_train, Y_test = train_test_split(features,target, test_size=0.25, random_state=42)
```

```
features.info()
```

```
#created values for alpha
alphas=10**np.linspace(5, -2, 15)
```

```
print(alphas)
```

```
# created another list and ran Ridge regression
#calculate the value for R2 and RMSE
rmse_sorted=[]
r2_sorted=[]
model_list=[]
for a in alphas:
    ridge = Ridge(alpha=a, max_iter=1000)
    ridge.fit(X_train, Y_train)
    pred = ridge.predict(X_test)
    r2=ridge.score(X_train,Y_train)
    rmse = mean_squared_error(Y_test, pred)**0.5;
    rmse_sorted.append(rmse)
    r2_sorted.append(r2)
    model_list.append(ridge)

print('RMSE:', rmse_sorted, '\nAlpha:', alphas, '\nR2:', r2_sorted)
```

```
#arranged it acc to the lest value for RMSE
alpha_index=np.argmin(rmse_sorted)
#got the index and saved it into a variable
best_alpha=alphas[alpha_index]
best_alpha
```

```
#appending to the list of step 5
fselection_list.append("Ridge, alpha ="+str(best_alpha))
ftransform_list.append("None")
rmse_list.append(rmse_sorted[alpha_index])
r2_list.append(r2_sorted[alpha_index])
```

```
#printing the list again
print(fselection_list)
print(ftransform_list)
print(r2_list)
```

7. Plot and summary analysis

```
#making a table using the values of list
result=np.vstack((fselection_list,ftransform_list,r2_list,rmse_list)).T
result_df=pd.DataFrame(result,columns=['Feature Selection','Feature Transform','R2','RMSE'])
result_df
```

```
#selecting the values bassed on the lowest value for RMSE
result_df.sort_values(by=['RMSE','R2'],ascending=[True,False]).head(1)
```

```
#running the query to make the best regression model and make predictions
features=df_selkBest
%run functions/Functions_KM146384.ipynb
transformed_df=make_poly(features)
X_train, X_test, Y_train, Y_test = train_test_split(transformed_df,target, test_size=0.25, random_state=42)
 # Create a scaler instance
scaler = StandardScaler()
    # Fit the scaler on the training data
scaled_train_X=scaler.fit_transform(X_train)
scaled_test_X =scaler.fit_transform(X_test)

    #remake the dataframe
X_train_scaled=pd.DataFrame(scaled_train_X,columns=X_train.columns)
X_test_scaled=pd.DataFrame(scaled_train_X,columns=X_test.columns)

# build the model
model = LinearRegression()
model.fit(X_train, Y_train)
# get the predictions for the test dataset
Y_pred = model.predict(X_test)
```

```
#printing the prediction
print(Y_pred)
```

```
transformed_df.head()
```

```
# Print the coefficients of the best linear model
print(pd.Series(model.coef_, index=transformed_df.columns))
```

```
#scatter plot using Y-pred and Y_test
plt.scatter(Y_pred,Y_test)
```

8. Out of Sample Prediction

```
df_selkBest.head()
```

```
df_selkBest.describe()
```

```
synthetic_data = df_selkBest.mean().to_frame().T
synthetic_data = df.quantile(0.05).to_frame().T
synthetic_data = df.quantile(0.75).to_frame().T
# Example selecting rows 0 and 2
synthetic_data = df.iloc[[0, 2]]
```

```
#creating a datframe
synthetic_data=pd.DataFrame(synthetic_data, columns=df_se
```

```
synthetic_data.head()
```

```
# Apply the same feature transformation as the best linea
poly_transform = make_poly(synthetic_data)
```

```
# Apply the scaling as the best linear regression model
scaler = StandardScaler()
    # Fit the scaler on the training data
scaled_train_X=scaler.fit_transform(poly_transform)
```

```
# Make predictions on the synthetic dataset
y_synthetic_pred = model.predict(scaled_train_X)

# Print the predictions
print("Predictions on Synthetic Dataset:")
print(y_synthetic_pred)
```

```
def select_best(data,num):
    kbest = SelectKBest(score_func=f_regression, k=num) # select 9 features
    X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

    kbest.fit_transform(X_train, y_train)
    df_kbest = data.iloc[:,kbest.get_support()]

    return df_kbest
```

```
def select_threshold(data):
    df_corr=df_copy.corr().abs()
    # Select the features with correlation value > 0.2
    selected_feature=df_corr['log_inst_review'].sort_values(ascending=False)
    selected_feature = selected_feature[selected_feature>0.2].index
    df_correlation = data[selected_feature].copy()
    # droping the column from selected features
    df_correlation.drop('log_inst_review',axis=1, inplace=True)
    return df_correlation
```

```
def selected_variance(data, threshold):
    #Select Features
    vt=VarianceThreshold(threshold)
    vt.fit(data)
    pd.DataFrame({'Variance':vt.variances_,'select_status':vt.get_support()},index=data.columns
    df_variance=data.iloc[:,vt.get_support()]
    return df_variance
```

```
def make_poly(data):
    poly_features = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
    transformed_data = poly_features.fit_transform(data)
    transformed_columns = poly_features.get_feature_names_out(data.columns)
    transformed_df = pd.DataFrame(transformed_data, columns=transformed_columns)
    return transformed_df
```

```
def select_threholdbonus(data,target):
    selected_feature=df_bonas.corrwith(target).abs().sort_values(ascending=False)
    selected_feature = selected_feature[selected_feature>0.2].index
    df_correlation = df_bonas[selected_feature].copy()
    return df_correlation
```

5.4 Feature Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
```

DO NOT SCALE the target variable

5.4.1 Standard Scaler

The standard scaler tries to standardize each value in a feature by removing the mean and scaling the varia

```
ss = StandardScaler()
scaled_features = ss.fit_transform(df_manual)

# you need to remake the dataframe
scaled_df1 = pd.DataFrame(scaled_features, columns=df_manual.columns)
scaled_df1
```

5.4.2 MinMax Scaler

In MinMax Scaling, the features will be scaled into range [0, 1], diving it by the range between the Max(x) a
and lower bound using feature_range variable.

```
ms = MinMaxScaler()
scaled_features = ms.fit_transform(df_manual)

# you need to remake the dataframe
scaled_df2 = pd.DataFrame(scaled_features, columns=df_manual.columns)
scaled_df2
```

5.4.3 Robust Scaler

The robust scaler tries to remove the disadvantage of the MinMax scaler such that it will not be affected by t
measure known as inter-quartile-range(IQR) that is the range between the 75% percentile and 25% percent

```
rs = RobustScaler()
scaled_features = rs.fit_transform(df_manual)

# you need to remake the dataframe
scaled_df3 = pd.DataFrame(scaled_features, columns=df_manual.columns)
scaled_df3
```

Multicollinearity analysis

We need to make sure that none of the features have a high multicollinearity among themselves to have a

```
df_independent = df.select_dtypes(include=np.number).drop('price', axis=1)

# create a dataframe for VIF data
vif_data = pd.DataFrame()
vif_data["feature"] = df_independent.columns

vif_data["vif"] = [variance_inflation_factor(df_independent.values, i)
                   for i in range(len(df_independent.columns))
]

print(vif_data)
```

```
scaled_features = StandardScaler().fit_transform(df_independent)
scaled_df1 = pd.DataFrame(scaled_features, columns=df_independent.columns)

# create a dataframe for VIF data
vif_data = pd.DataFrame()
vif_data["feature"] = scaled_df1.columns

vif_data["vif"] = [variance_inflation_factor(scaled_df1.values, i)
                   for i in range(len(scaled_df1.columns))
]

print(vif_data)
```

```
scaled_df1.drop('tax', axis=1, inplace=True)
# create a dataframe for VIF data
vif_data = pd.DataFrame()
vif_data["feature"] = scaled_df1.columns

vif_data["vif"] = [variance_inflation_factor(scaled_df1.values, i)
                   for i in range(len(scaled_df1.columns))
]

print(vif_data)
```

```
df.drop('tax', axis=1, inplace=True)
```

Feature Selection, Feature Transformation and Feature Scaling

You may want to perform feature selection, feature transformation (poly, log1p, etc) or feature scaling before making the model.

Split the dataset into training and test sets

We will split 80:20 and using randomstate=42 to make a repeatable result. You may want to shuffl

```
# first let divide the df into features and response
response = df.price
features = df.drop('price', axis=1)
```

```
# import 'train_test_split'
X_train, X_test, Y_train, Y_test = train_test_split(features, response, test_size=0.2, rando
```

Training and Testing

```
# build the model
model = LinearRegression()
model.fit(X_train, Y_train)
```

```
# get the predictions for the test dataset
Y_pred = model.predict(X_test)
```

Calculate the performance metric

```

```

```
# R-squared
r2 = model.score(X_test, Y_test)
print("R2 score is ", r2)
```

```
from sklearn.metrics import mean_squared_error
# RMSE
rmse = mean_squared_error(Y_test, Y_pred, squared=False)
print("RMSE is ", rmse)
```

```
# plot the prediction vs actual
plt.scatter(Y_pred, Y_test, alpha=0.7)
plt.title('Linear Regression Model')
plt.xlabel('Predicted Value')
plt.ylabel('Actual Value')
plt.show()
```

```
# the coefficient list
feat = list(X_train.columns)
coef = model.coef_.transpose()
pd.DataFrame(zip(feat,coef), columns=['Feature', 'Coefficients'])
```

Ridge Regression

```
# import the libraries
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# load the dataset
df = pd.read_csv("boston_modified.csv")
df.head()

# just drop all NAN
df.dropna(inplace=True)

# just transform all categorical
# we need to be sure that all the NAN are removed and all other columns are numerical
df=pd.get_dummies(df, drop_first=True)
```

```
df.head()
```

```
target = df.medv * 25000 # you still remember why we need to multiply by 25000 rite?
features = df.drop('medv', axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.25)
```

The Ridge() function has an alpha argument that is used to tune the model. We'll generate an array of alpha values ra
essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares f

```
alphas = 10**np.linspace(5, -2, 15)
alphas
```

```
# let's ridge them!
rmse_list = []
r2_list = []
model_list = []

for a in alphas:
    ridge = Ridge(alpha=a, max_iter=1000)
    ridge.fit(X_train, Y_train)
    pred = ridge.predict(X_test)

    r2_list.append(ridge.score(X_train,Y_train))
    rmse = mean_squared_error(Y_test, pred)**0.5;
    rmse_list.append(rmse)
    model_list.append(ridge)
    print("Alpha",a,"RMSE",rmse)

ridge_result = np.vstack((alphas, rmse_list, r2_list)).T
ridge_df = pd.DataFrame(ridge_result, columns=['Alpha', 'RMSE', "R2"])
ridge_df
```

```
ridge_df.sort_values(by=['RMSE', 'R2'], ascending=[True, False]).head(1)
```

```
# find out the coefficient for the best result
best_model = model_list[14]
```

```
print(pd.Series(best_model.coef_, index=features.columns))
```

Univariate Analysis

In univariate analysis, we try to see the distribution of some interesting feature. We can use histogram, distplot, bar chart, boxplot or violin plot for this purpose

Lasso Regression

The Lasso() function has an alpha argument that is used to tune the model. We'll generate an array of alpha values ranging 1 to a small value:

```
alphas = np.linspace(1, 0.001, 15)
alphas
```

We will use the same dataset

```
xtrain, xtest, ytrain, ytest = train_test_split(features, target, test_size=0.25)
```

```
# Let's put them in the Lasso! yee haw
rmse_list = []
r2_list = []
model_list = []

for a in alphas:
    lasso = Lasso(alpha=a, max_iter=1000)
    lasso.fit(X_train, Y_train)
    pred = lasso.predict(X_test)

    r2_list.append(lasso.score(X_train,Y_train))
    rmse = mean_squared_error(Y_test, pred)**0.5;
    rmse_list.append(rmse)
    model_list.append(lasso)
    print("Alpha",a,"RMSE",rmse)


lasso_df = pd.DataFrame(zip(alphas, rmse_list, r2_list), columns=['Alpha', 'RMSE', "R2"])
lasso_df
```

```
lasso_df.sort_values(by=['RMSE', 'R2'], ascending=[True, False]).head(1)
```

```
# find out the coefficient of the best result obtained from our lasso
best_model = model_list[14]
print(pd.Series(best_model.coef_, index=features.columns))
```

```
# Assigning multiple variables can be done simultaneously
x, y, z = 10, 2.5, "hello"; text = " world"
print(x)
print(y,z)
print(z + text)
```

Numbers

- Python support integer and floating numbers

```
# number data type exercise
mydata = 10
print(mydata)

mydata = 112.345
print(mydata)

myNextData = float(10)
print(myNextData)
```

```
# The cell below is needed to style the table to be displayed on the left
ft, but it may destroy the layout of the cell
```

while loops

- A while loop specifies a condition ar

```
# example of while loop
x = 256
total = 0
while total < 500:
    total += x
    x = x // 2

print(total)
```

for loops

- for loops are for iterating over a collection (like a li
- The standard syntax for a for loop is:

```
# example of for loop
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total = 0
for val in sequence:
    total += val

print(total)
```

```
# another example of foor loop
data = ["hello", 1, -200.5]
for value in data:
    print(value)
```

```
# an example of using range or len
data = ['marry', 'has', 'a', 'little', 'lamb']
for i in range(len(data)):
    print(i, data[i])

print()

for i,val in enumerate(data):
    print(i, val)
```

and Continue

e break statement stops look execution while the conti
nested loops, the break statement can be used to brea

```
# example of break and continue
for n in range(2,10):
    isPrime = 1

    for x in range(2,n):
        if n%x == 0:
            print(n, 'equals', x, '*', n/
            isPrime=0
            break

    if isPrime>0:
        print(n, 'is a prime number')
```

2. Lists

- Python's lists are the most flexible data type. It can be created t
need not be of the same data type
- It is similar to array in JS. It can hold as many member as you v

```
# Creating and populating list
myList = [] # an empty list
myList.append('CS153290')
myList.append("52023")
myList.append('001')
print(myList)


alist = list(range(10))
print(alist)
for val in alist: print(val)
```

```
# Be corefult with the index
myList[3]
```

```
# to find out the length of a list you need to
print(len(myList))
```

```
# distribution plot
sns.displot(tips.tip, color='g',bins=20)
```

```
indexToDrop = tips[tips.tip>6].index
print(indexToDrop)
tips.drop(indexToDrop, inplace=True)
```

```
sns.displot(tips.tip, color='g',bins=10)
```

```
# distribution plot
from scipy.stats import norm
sns.distplot(tips.tip, fit=norm)
plt.show()
sns.histplot(tips.tip, kde=True)
```

```
sns.countplot(data=tips, x="day")
```

```
tips.day.value_counts().plot(kind='bar', color=list('bygr'))
```

1. Tuple

- Almost similar to list, however instead of square bracket, it uses parenthesis
- Tuple is also immutable, i.e., cannot be changed

```
myTup = () # an empty tuple

myTup = (10,) # try to remove the comma
print(myTup)
print(myTup[0])

myTup = ('a', 'x', 4, -5)
print(myTup[1])
print(myTup[-3])
#myTup[0] = 'hello' # you will get an error with this stat

myTup = tuple('hello123')
print(myTup)
```

```
tup = tuple((True, [3,7], 'orange'))
print(tup[1])
tup[1][0] = 5
print(tup[1])

tup = (5, 'banana', 0.12, -4, False)
```

1.1 Basic List operation

```
myList.extend(['N6107','Wed'])
print(myList)

num = [1, 2, 5, 1, 7, 1]
alist = [i for i in num if i>1] # list comphre
print(alist)

data = ['a', 'as', 'bat', 'car', 'python']
alist = [x.upper() for x in data if len(x)>2]
print(alist)
```

```
# zip: pairs up the elements in a number of li
# it is useful when we need to pair up parame
seq1 = ['Douglas', 'likes', 'Chocolate']
seq2 = [20, 30, 50]
zipped = zip(seq1, seq2)
print(list(zipped))

# enumerate: use in a for loop, it will enumer
# with the members of a sequence
for i,(a,b) in enumerate(zip(seq1,seq2)):
    print("{0}: {1} {2}".format(i,a,b))
```

4.2. Plotting from Pandas

```
# Note, you can also create subplot in Pandas

# Load the iris data available in sklearn
from sklearn.datasets import load_iris
import pandas as pd
iris=load_iris()

# Read sample data
df = pd.DataFrame(iris.data)
df.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
print(df)

# plot the dataframe
print("Histogram:\n")
df.hist()
plt.tight_layout()
plt.show()

print("Line chart:\n")
df.plot()
plt.show()

print("Box plot:\n")
df.boxplot()
plt.show()
```

```
index = ['one', 'two', 'three', 'four', 'five']
columns = ['A', 'B', 'C', 'D']
df = pd.DataFrame(abs(np.random.randn(5, 4)), index=index, columns=columns)

# matplotlib can be used to create heatmap table.
# A better heatmap with good color choice can be generated using Seaborn
# However, you can also style the dataframe to highlight its values
df.style.background_gradient(cmap='Blues', axis=None)
```

2. Function

```
# Simple function to add two numbers
def add_two_numbers(x,y):
    return x + y
print(add_two_numbers(10, -5))
```

```
# returning multiple values
def f():
    x, y, z = 5, 6, 7
    return x, (y, z)
first, (a, b) = f()
print(first, a, b)
```

```
# Variable Length Arguments
def accumulate(*args):
    total=0
    for x in args:
        total += x
    return total
data = range(1,6)
print("The data is ", list(data))
print("Total of the data is ", accumulate(*data
print("Total of the data is ", accumulate(*data
```

2.3 "Args, **Kwargs and additional Ipython file

```
%run Function/myfunction.ipynb
```

```
data = [3, 2, 1]
myTotalSum(*data)
```

```
params = {}
params['base'] = 10
params['height'] = 12
myAreaCal(**params)
```

2.4 Create and use your own module

```
from Module import naming_generator as ng

name = ng.generate()
```

2.1. Array initialization and creation

```
#Example code for initializing Numpy array
a = np.array([1, 5, 7, 9])
print(a)
print(type(a))

print(a[2], a[-1])
```

```
# Create a rank 2 array
b = np.array([[1,5,5],[2,4,8]])
print(b)
print(b.shape)
```

```
# Create a 2x3 array of all zeros
c = np.zeros((2,3))
print(c)
# there is also ones(), full(), eye(). Look t
```

```
# arange() will create arrays with regularly
d = np.arange(2,8)
print(d)
d = d.reshape((2,3))
print(d)
```

```
# type casting
print(d.dtype) # data type
d_float = d.astype(np.float64)
print(d_float)
```

2.2. Random, LinSpace

```
# numpy random.randint can be used to generat
e = np.random.randint(7)
print(e)
e = np.random.randint(1,7,size=10)
print(e)
```

```
# Create a 3x3 array filled with random value
f = np.random.random((3,3))
print(f)
```

```
# Linspace() will create arrays with a specif
g = np.linspace(1, 1000, 6)
print(g)
```

5. Feature Engineering

5.1.1. Rounding

When we deal with percentage values, we may not need the values in a high amount precision. So, we can round them up to the nearest 10, 100, etc.

```
df = pd.read_csv("item_popularity.csv")
df.head()
```

```
df['pop_scale10'] = np.array(np.round((df['pop_percent'] * 10)), dtype='int')
df['pop_scale100'] = np.array(np.round((df['pop_percent'] * 100)), dtype='int')
df
```

Diction

```
stu = {} # an empty dictionary
stu['Douglas'] = '1234'
stu['John'] = '3456'
stu['Jack'] = '5678'

print(stu)
for k,v in stu.items():
    print('Student ID of {0} is {1}'.format(k,v))

stu = {'Douglas':[100, 80,55], 'John':'B', 'Jack':'A-'}
print(stu)
```

```
# operation
seq1 = ['one', 'two', 'three']
seq2 = reversed(range(len(seq1))) # [2, 1, 0]
mapping = dict(zip(seq1, seq2))
print(mapping)
```

```
# Import seaborn
import seaborn as sns

# Load an example dataset
tips = sns.load_dataset("tips")

print(tips)

# relplot shows relationship between variables
sns.relplot(
    data=tips,
    x="total_bill", y="tip", col="time", hue="smoker", size="size"
)
```

```
# Basic slicing : The basic slice syntax is i: j: k
# where i is the start index,j is the stopp index,
a = np.array([5, 6, 7, 8, 9])
print(a[1:7:2])
print(a[-2:5])
print(a[-1:1:-1])
```

```
#Transpose function
a = np.array([1, 2, 3, 4, 5, 6])
a = a.reshape((3,2))
print(a)
print(a.T)
```

```
# example boolean operation
a = np.random.randint(1, 5, size=6).reshape((2,3))
print(a)
print(a > 2)
print(a[a>2])
```

```
# statistics
print(a.sum())
print(a.mean())
print(a.mean(axis=0)) # axis=0 is row, expecting op
print(a.mean(axis=1))
```

3.1.1 Pandas Series

```
#Creating a pandas series
# creating a series by passing a list of values, and a custom index lab
#Note that the labeled index reference for each row and it can have
#duplicate values
s = pd.Series([1, 3, 5, np.nan, 5, 6], index=['a','b','c','d','e','f'])
print(s)
print(s.index)
print(s['b'])
```

```
# example of creating series
data = {'banana': 5, 'apple': 2, 'kiwi':8}
s = pd.Series(data)
print(s)
fruit = ['banana', 'apple', 'grape']
s = pd.Series(data, index=fruit)
print(s)
```

```
# checking for nulk or NA
s.isnull()
s.isnull().sum()
```

3.1.1 Pandas Dataframe

```
#Creating a pandas dataframe
data = {
    'marriage status': ['M','S', 'S'],
    'ID' : ['e1', 'e2', 'e3'],
    'age': [30, 25, 40]
}

df = pd.DataFrame(data)
print(df)
#df = pd.DataFrame(data, columns=['ID', 'age', 'marriage status'])
#print(df)
print(df.ID)
print(df['marriage status'])
#print(type(df.ID))

# we cannot use the dot df. below because the column name is not valid
#print(df['marriage status'])

print(df.columns)
df.columns = ['marital_status','id','age']
print(df)
print(df.marital_status)
```

```
# creating dataframe from a dictionary
data = {
    'fruit': ['Banana','Apple', 'Kiwi'],
    'amount' : [100, 30, 150]
}
df = pd.DataFrame(data)
print(df)
df.columns = ['FRUIT','INVENTORY']
print(df)

# create a dataframe from numpy array
# use reshape() method of np
# use arange() function: similar to range() function
df = pd.DataFrame(np.arange(15).reshape(5,3), columns=['a','b','c'])
df
```

```
#application when you want to perform some transformation based on the
#example
df_t = pd.DataFrame({'food':['Pastrami','corned beef', 'Peking duck',
                             'pastrami', 'nova lox', 'Foie gras'],
                     'ounces':[6,7.5,5,3,6,1.5]})

print(df_t)

#suppose you want to add a column indicating the type of animal that
#of each distinct meat type to the kind of animal where it comme from:

map_meat_to_animal={
    'pastrami' : 'cow',
    'corned beef' : 'cow',
    'nova lox': 'salmon',
    'foie gras': 'duck',
    'Peking duck': 'duck'
}

df_t["animal"] = df_t.food.map(map_meat_to_animal)
print(df_t) # you will get NaN values because the map and df_t.food ho
```

naming Axis Indexes

```
# generate a dataframe
data = pd.DataFrame(np.arange(6).reshape(3,2),
                    index=['Vancouver','New Westminster','Richmond'],
                    columns=['one','two'])
print(data,"\n")
# replace the index to the first four char of the original index and m
data.index = data.index.map(lambda x: x[:4].upper())
print(data,"\n")
```

GroupBy and Transform

```
# generate some data
user = np.random.randint(101, 105, 12)
user = np.sort(user)
purchase = np.random.randint(10, 20, 12)*25
data = np.vstack((user, purchase)).T # combine the series of user and p

# create a dataframe
df = pd.DataFrame(data, columns=['userID','purchase'])
print(df,"\n")

# use groupby to find the mean average purchase per user
mean = df.groupby('userID')['purchase'].mean()
print(mean,"\n")

# if you want to create a new column with groupby result, you need to
# The transform function retains the same number of items as the origi
# transform() calls func on self producing a DataFrame with transforme
# Produced DataFrame will have same axis length as self.
df["user_mean"] = df.groupby('userID')['purchase'].transform('mean')
print(df,"\n")
```

```
plt.bar(x,y)
plt.show()
plt.plot(x,y)
plt.show()
plt.scatter(x,y)
```

```
# generate some data to plot
data1 = np.random.randn(10).cumsum()
data2 = np.random.randn(10).cumsum()
data3 = np.random.randn(10).cumsum()

data4 = np.random.randn(100)
# subplot
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(223)

# matplotlib will draw on the last figure object
# K-- means black dash, go- means green with marker and line
plt.plot(data1, 'k--')
plt.plot(data2, 'go-')
plt.plot(data3, color='b', linestyle='-', marker='s')

# alpha is transparancy
ax1.hist(data4, bins=10, color='g', alpha=0.2)

ax2.scatter(np.arange(30), np.arange(30) + 3*np.random.randn(30))
ax2.set_title('scatter plot')
ax1.set_title('histogram')

# adjusting the subplot space. values are in percentage of the fig
#subplots_adjust(left=None, bottom=None, right=None, top=None, ws
plt.subplots_adjust(wspace=0.4, hspace=0.4)
```

```
from numpy.random import randn
# generate data
data1 = randn(100).cumsum()
data2 = randn(100).cumsum()
data3 = randn(100).cumsum()

# you can use figsize to guarantee the figure has a certain size
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(111)
ax.plot(data1, 'k', label='one')
ax.plot(data2, 'g--', label='two')
ax.plot(data3, 'b.', label='three')
ax.legend(loc='best')

ax.set_title('Plot of some random numbers')
ax.set_xlabel('X values')
ax.set_ylabel('Y values')
ax.set_xlim([40, 100])

# saving the figure
plt.savefig('myplot.png', dpi=400, bbox_inches='tight')
```

3.2.1 Selecting and Indexing

```
# create a dataframe
data = {'country': ['Canada', 'USA', 'Brazil'],
        'capital': ['Ottawa', 'Washington', 'Brasilia']
}
df = pd.DataFrame(data)
df
```

```
# Indexing, you can use
# position with iloc or 1st, for example iloc[[0],[0]] or 1st[[0],[0]]
# Label with the column name, and combination of label and position. for
# loc[[0],['country']] or at[[0], 'country']
country = df.iloc[[0],['country','capital']]
print(country,'\n')
country = df.loc[[0],:]
print(country, '\n')
country = df.loc[:,['country']]  # df.country
print(country,'\n')
```

```
# create a dataframe
df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f', 'h'],
                  columns=['one', 'two', 'three'])
print(df)
print(df.index)
print(df.columns)
```

```
# reindex
df2 = df.reindex(['a','b','c','d','e','f','g','h']) # taking subset
print(df2)

df.index = ['d1','d2','d3','d4','d5']
print(df)
```

cov() - Covariance

```
df.cov()
```

corr() - Correlation

```
df.corr()
```

```
df.corr()["median_house_value"].abs().sort_values(ascending=False)
```

3.2.2 Arithmetic

```
df1 = pd.DataFrame(np.arange(12).reshape((3,4)), columns=list('abcd'))
df2 = pd.DataFrame(np.arange(20).reshape((4,5)), columns=list('abcde'))

print(df1,"\n")
print(df2,"\n")

df2.loc[1,'b'] = np.nan
print(df2,"\n")

df3 = df1+df2
print(df3,"\n")

print(df3.isnull().sum()) # you can specify axis=1 in sum() if you want

df3 = df1.add(df2, fill_value=0)
print(df3)
```

3.2.3 Apply and Map

```
df = pd.DataFrame(np.random.randn(4,3), columns=['a','b','c'])
print(df)

f = lambda x: x.max() - x.min()
df.apply(f) # apply by default will be per columns

df.apply(lambda x: x.max() - x.min(), axis='columns') # apply to axis co

#df.columns = ['A','B','C']
#df.columns = [x.upper() for x in df.columns]
df.columns = df.columns.astype(str).str.upper()
print(df,"\n")
```

ndling Categorical Data

abel Encoder

```
# drop the unneeded column
#df.drop(['#'], axis=1, inplace=True)

from sklearn.preprocessing import LabelEncoder

# resample the data to get all the distinct values displayed at the top and rest
df2 = df.sample(random_state=1, frac=1).reset_index(drop=True)

gle = LabelEncoder()
gen_label = gle.fit_transform(df2.Generation)
#gen_label
gen_mapping = {index: label for index, label in enumerate(gle.classes_)}
#gen_mapping
df2['GenerationLabel'] = gen_label
df2.head(10)
#np.unique(df.Generation)
```

ne hot Encoding

```
# one hot encoding transform one feature into n binary features
from sklearn.preprocessing import OneHotEncoder
gen_ohe = OneHotEncoder()
gen_feature_arr = gen_ohe.fit_transform(df2[['GenerationLabel']]).toarray()

# we need the labels from the labelEncoder, or we can put it ourselves
#gen_features = pd.DataFrame(gen_feature_arr, columns=['Gen 1', 'Gen 2', 'Gen 3
df3 = df2[['Name', 'Generation']].join(gen_features)
df3
```

ummy Features

```
# dummy coding transform one feature into n-1 feature
dummy_features = pd.get_dummies(df2.Generation, drop_first=True)
df3 = pd.concat([df2[['Name', 'Generation']], dummy_features], axis=1)
df3
```

ving Duplicates

```
#use drop_duplicates() method
df.duplicated(['Customer'])
df1 = df.drop_duplicates(['Customer'])
print(df1)
```

string manipulation

```
# replace column names to be lower case
df2 = df1.copy() # copy the dataframe to df2
df2.columns = df.columns.str.lower()
print(df2,"\n")

# remove whitespace in column customer. You may get a warning message
df2.customer = df2.customer.str.replace(" ","")
print(df2,"\n")
```

```
# uniq and value counts
df2.type.unique()
```

```
# replace some values that contains certain string

#df2.loc['type'] = 'online' # (?i) means I don't care about the case letter

df2.loc[df2.type.str.contains('(?i)online'), 'type'] = 'online'
df2.loc[df2.type.str.contains('(?i)other|store'), 'type'] = 'other'
print(df2)
df2.type.value_counts()
```

```
# reorder the column
df2 = df2[['customer','type','purchase']]
df2
```

```
# find rows where purchase is greater than 1
df2[df2.purchase>1]
df2[df2.purchase>1].index # get the index

# find rows where purchase is less than 5 and type is other
```

5.1.3 Transformation: log transform

Log transforms are useful when applied to skewed distributions as they tend to expand the values which compress or reduce the values which fall in the range of higher magnitudes. This tends to make the ske

Note: for data that may contain zero values, instead of using np.log, we can use np.log1p instead. np.lo data

```
att_def_log = att_def.copy()
att_def_log['logio_attack'] = att_def.Attack.apply(lambda x: np.log1p(x))
att_def_log['logio_defense'] = att_def.Defense.apply(lambda x: np.log1p(x))

#att_def_log['logio_defense'] = np.log1p(att_def.Defense)
```

```
att_def_log.head()
```

```
# selecting
print(df2.head(3),"\n")

print(df2.tail(3),"\n")

# there's also functions like nlargest, nsmallest
print(df2.nlargest(2,'one'),"\n")
print(df2.nsmallest(1,'two'),"\n")
```

```
data = {
    'name' : ['jack', 'john', 'doc', 'sam'],
    'dept' : ['computing', 'science', 'computing', 'science'],
    'test1': [np.random.randint(40,80) for i in range(4)],
    'test2': np.random.randint(40,90, size=4)
}

df = pd.DataFrame(data)
print(df,"\n")

print(df.query('dept == "computing" and test1<70'),"\n")

print(df[(df.test1>50) & (df.test2>50)],"\n")

print(df[df.name > 'jack'], "\n")

print(df[df.name.isin(['john','doc'])],"\n")

print(df[~df.name.isin(['john','doc'])],"\n")

# df.filter is to filter row or column
print(df.filter(items=['dept']),"\n")

print(df.filter(like='test', axis=1),"\n")
```

3.5. Basic Statistics

```
# load the data
df = pd.read_csv('housing.csv')
df.head()
```

```
df.info()
```

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

```
df.dropna(inplace=True)
df.isnull().sum().sum()
```

```
# prepare the dataframe and get
df.info()
df.describe()
```

Filling Values

```
df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns=list('ABCD'))
df
```

```
fill = {'A':0, 'B':1, 'C':2, 'D':3}
df.fillna(value=fill)

df.fillna(value=fill,inplace=True) # make the f
print(df)
```

```
# another example
# create a dataframe with some nan values
df = pd.DataFrame([
    [np.nan, 2, np.nan, 0],
    [3, 4, np.nan, 1],
    [np.nan, np.nan, np.nan, 5],
    [np.nan, 2, np.nan, 4],
],
    columns=list('ABCD')
)

print(df)
```

```
# AFAIK fillna with axis is buggy, especially if you want t
def fill_mean(x, opwise):
    return x.fillna(x.mean(), axis=opwise)

fill_mean(df,0)
```

```
# If you want to fill column-by-column with the row average,
# use lambda and apply
df2 = df.apply(lambda row: row.fillna(row.mean()), axis=1)
print(df2)
```

```
# you can fill the nan with the value from other column
print(df,"\n")

# fill the NaN in A with 1.5 of D
df.A.fillna(1.5*df.D, inplace=True)
print(df,"\n")
```

```
# dropping values in Series

# note: NaT represents missing value for datetime
df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
                   "toy": [np.nan, 'Batmobile', 'Bullwhip'],
                   "born": [pd.NaT, pd.Timestamp("1940-04-25"),
                            pd.NaT]})
df

# you can drop values with drop(). By default, drop is on axis=0
df.drop(1)
df.drop('name',axis=1) # without inplace the drop is not permanen
print(df,"\n")

df.drop(columns={'name','born'},axis=1, inplace=True)
print(df,"\n")

df = pd.DataFrame(np.random.randint(1,10, size=20).reshape(4,5),
                  columns=['a','b','c','d','e']
                  )

# inject some nan
for ind in df.index:
    df.loc[ind, df.sample(frac=0.5, axis=1).columns] = np.nan

print(df,"\n")

df.isna()
df.dropna(axis=1)
df.dropna(thresh=3, axis=1)
```