Background Information

Food insecurity is a problem plaguing a large proportion of the world's population.

With about **8 million adults** and **5 million children** living in food insecure households. There are a multitude of factors leading to the existence of food insecurity; poverty and monetary reasons being the prominent causes.

Fruits and Vegetables are generally the first food groups of food people forgo as financial hardships hit. Rather, people choose to go for staple foods such as rice and beans in order to feel satiated for longer. By doing this, they lose out on many of the essential nutrients that are present in fruits and vegetables.

User persona chosen - Government

About
The exercise of authority over a state, district, organization, institution

Goals
End hunger, achieve food security and improved nutrition

Ability
Access to new and current data
Exert price control on the market
Manage imports and exports for food security
Distribution of food to ensure food security

How we came up with the model

Knowing price index allows governments to **plan and implement measures** to ensure all citizens are **able to afford fruits and vegetables** necessary for a healthy and active lifestyle. (Goal - achieve food security, improve nutrition)

If the price index exceeds government expectations, they can identify the factor that causes the price index to rise disproportionately and implement measures to **mitigate the factor's effect on price index of fruits and vegetables**, increasing food security of a nation. (Ability of government)

PROBLEM STATEMENT:

How might we predict the price index of fruits and vegetables using different factors of supply?
I.e Population, Arable land, Production Capacity, Fruits and Vegetables Imports and GDP per Capita

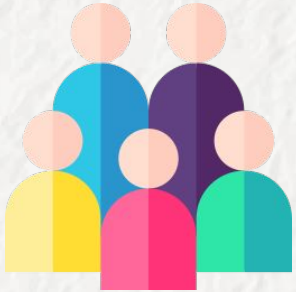
HYPOTHESIS:

An increase in supply of fruits and vegetables due to different factors of supply would result in a lower price index of fruits and vegetables.

Factors of Supply Chosen:

Population: With the increasing population worldwide, the demand for food is projected to double in the years to come which supply is unable to meet. As a result, price index of fruits and vegetables, will increase and cause food insecurity in countries with large populations.

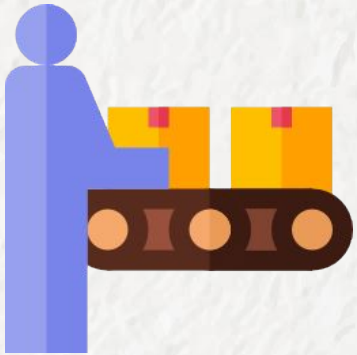
Arable land: Arable land is declining due to industrialisation and urbanisation, decreasing supply of food. Assuming demand remains constant and price index of food increases as quantity of fruit and vegetables demanded increases, decreasing food security in a country as fewer consumers can afford fruits and vegetables.



Factors of Supply Chosen:

Production Capacity: Increased production capacity increases supply of food decreases price index.

Fruit and Vegetable Imports: Governments import fruits and vegetables to ensure sufficient amount of food for the population. As amount of imported food increases, overall supply of food will increase, causing a decrease in price index.



Factors of Supply Chosen:

GDP per Capita: Increased GDP Per Capita indicates more disposable income for citizens of a country, granting them higher purchasing power. Citizens no longer experience food insecurity because of higher purchasing power as they can purchase larger amounts of food than previously, increasing the demand for fruits and vegetables.

Data choice and preparation

- **Why did we choose this data set?**
  - The data set we chose was from 2017. We chose factors we believed would make the most impact on the price indexes of fruits and vegetables in this globalised, industrialised world
- **Why and how did we clean up data**
  - There were some unnamed columns as well as some columns that were not needed for this model.
  - We removed these columns and data sets using drop() function.
  - We also removed data points we believed were outliers as they produced a heavily skewed predicted dataset.
- **Splitting data 70%, 30% randomly**
  - 70% is for training the model to increase accuracy and prevent any biases and overfitting
  - 30% is for testing the model to ensure the model predicts the price index according to our equation

```
In [10]: #Removing any extra columns not needed for this model:
         df=df.drop(["Unnamed: 25", "Unnamed: 26", "Unnamed: 27", "Unnamed: 28","Unnamed: 29", "Unnamed: 30", "Unnamed: 31", "Unnamed: 32"]
```

Explanation of model

- **Discussion on predictors or features**
  - **Predictor(Target):** Price Index of fruits and vegetables across different countries
  - **Features:** Population, Arable Land/Acres, Veg and fruits gross production/1000USD

```
columns_features=["Population","Arable Land/Acres","Veg and fruits gross Production value/1000USD"]
```

Column used for target - Price index of fruits and vegetables across different countries

Rationale for choosing price index of fruits and vegetables: Using the price index of fruits and vegetables allows us to analyse the various reasons for the outcome of the model.

```
# PREPARING DF_FEATURES AND DF_TARGET:

#This function allows to convert the dataset of features and target into dataframes
def get_features_targets(df, feature_names, target_names):
    df_feature=pd.DataFrame(df[feature_names])
    df_target=pd.DataFrame(df[target_names])
    return df_feature, df_target

#preparing df_feature and df_target
df_features,df_target=get_features_targets(df,columns_features,["Price Index Total"])
```

Explanation of model

- **Code and discussion on building the mode**
  - Splitting the data 70% for training, 30% for testing

```
#splitting data into training and test:

#function to split data with random_state=100 and test_size=0.3:
def split_data(df_feature, df_target, random_state=None, test_size=0.5):
    indexes = df_feature.index
    if random_state != None:
        np.random.seed(random_state)
    k = int(test_size * len(indexes)) # 0.3 * 506 --> 151
    test_index = np.random.choice(indexes, k, replace=False)
    indexes = set(indexes)
    test_index = set(test_index)
    train_index = indexes - test_index # set subtraction

    df_feature_train = df_feature.loc[train_index, :] # select the rows as per train_index, and all its columns
    df_feature_test = df_feature.loc[test_index, :]
    df_target_train = df_target.loc[train_index, :]
    df_target_test = df_target.loc[test_index, :]
    return df_feature_train, df_feature_test, df_target_train, df_target_test

df_features_train,df_features_test,df_target_train,df_target_test=split_data(df_features,df_target,random_state=100,test_size=0.3
```

Explanation of model

- Code and discussion on building the mode
  - Normalization of dataframe, conversion of feature and target training dataframe

```
#function for Z normalization:
def normalize_z(df):
    dfout=df.copy()
    dfout=(df-df.mean(axis=0))/df.std(axis=0)
    return dfout

df_features_train_z=normalize_z(df_features_train)
```

```
#prepare_feature function converts the feature training dataframe into a numpy array, change into a column vector and add a colum
def prepare_feature(df_feature):
    cols=len(df_feature.columns)
    feature=df_feature.to_numpy().reshape(-1,cols)
    X=np.concatenate((np.ones((feature.shape[0],1)),feature),axis=1)
    return X

X = prepare_feature(df_features_train_z)

#prepare_target function converts the target training dataframe into a numpy array and change into a column vector
def prepare_target(df_target):
    cols=len(df_target.columns)
    target=df_target.to_numpy().reshape(-1,cols)
    return target

target = prepare_target(df_target_train)
```

Explanation of model

- Code and discussion on building the mode
  - Calculating gradient descent using calculating linear and computing cost function

```
#calculating the gradient descent using calc_linear and compute cost functions
def calc_linear(X, beta):
    return np.matmul(X,beta)
    pass

def compute_cost(X, y, beta):
    J = 0
    m=X.shape[0]
    error=calc_linear(X,beta) - y
    error_sq=np.matmul(error.T,error)
    J=(1/(2*m))*error_sq
    J=J[0][0]
    return J

def gradient_descent(X, y, beta, alpha, num_iters):
    m=X.shape[0]
    J_storage=np.zeros((num_iters,1))
    for n in range(num_iters):
        deriv=np.matmul(X.T,(calc_linear(X,beta)-y))
        beta=beta-alpha * (1/m) * deriv
        J_storage[n]=compute_cost(X,y,beta)
    return beta, J_storage

iterations = 1500
alpha = 0.01
beta = np.zeros((4,1))

beta, J_storage = gradient_descent(X,target,beta,alpha,iterations)
```

Explanation of model

- Code and results of the metric on data set

```
#predict functions:
def predict_norm(X,beta):
    return np.matmul(X,beta)

def predict(df_feature, beta):
    feature_norm=normalize_z(df_feature)
    X=prepare_feature(feature_norm)
    return predict_norm(X,beta)

pred = predict(df_features_test,beta)

'''
Describing the pred dataset:
'''

#The datatype of predicted array:
print("Type of predicted array - ", type(pred))
print()

#The number of rows and columns in predicted array:
print("Shape of predicted array - ", pred.shape)
print()

#The mean of predicted array:
print("Mean of predicted array - ", pred.mean())
print()

#The standard deviation of predicted array:
print("Standard deviation of predicted array - ", pred.std())
```

```
Type of predicted array -  <class 'numpy.ndarray'>

Shape of predicted array -  (35, 1)

Mean of predicted array -  222.41969881407326

Standard deviation of predicted array -  25.608495453785885
```

Explanation of model

- Code and results of the metric on data set

Model Evaluation

INITIAL METRICS

r^2 value

r^2 is not a good metric for multiple linear regression because it always increases when we add a predictor variable

FINAL METRICS

Adjusted r^2 value:
compensates for the addition of variables and only increases if the new predictor enhances the model above what would be obtained by probability

Mean Squared Error (MSE)

check how close the estimates are to the actual values

Mean Squared Error (MSE)

Model Evaluation

Code and results of the metric on your test data set

```
#calculating MSE:
def mean_squared_error(target, pred):
    n=target.shape[0]
    diff=np.sum((target-pred)**2)
    return diff/n

target = prepare_target(df_target_test)
mse=mean_squared_error(target,pred)

print("Mean Squared Error is - ",mse)

#calculating adjusted r2 value:
def r2_score(y, yhat):
    R2=1 - np.sum((yhat - y)**2) / np.sum((y - np.mean(y))**2)
    return R2
#k = number of independent variables
def adjusted_r2(y,p,R2):
    n_hello=y.shape[0]
    adj_rsquared = 1 - (1 - R2) * ((n_hello - 1)/(n_hello-p-1))
    return adj_rsquared

# r2=r2_score(target,pred)

# print(r2)

adjusted_r2=adjusted_r2(target,3,r2)
print("adjusted r2 - ",adjusted_r2)
```

```
Mean Squared Error is -  4472.2241023635925
adjusted r2 -  -0.02874856178286933
```

Results & Improvements

Iteration 2:

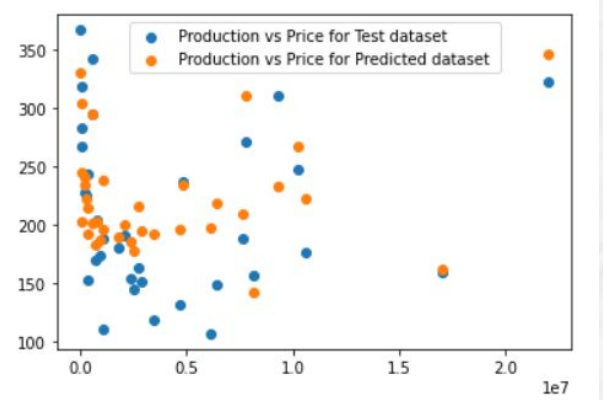
We added **more supply factors** affecting price of fruits and vegetables as we believed adding more factors could give us a more comprehensive and accurate prediction of price index for more countries. (GDP per Capita, Imports)

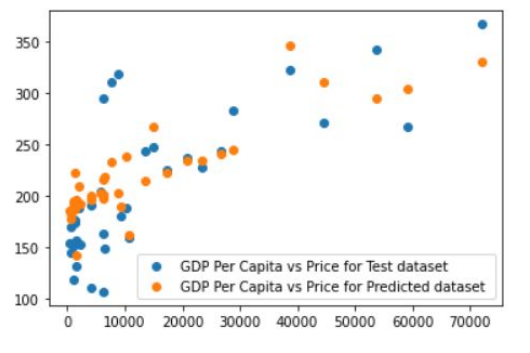
Iteration 3:

Applying natural logarithm to observe a more linear relationship between the variables and see the correlation between the independent and dependent variables.

```
EATURES:
olumns_features1=["Veg and fruits gross Production value/1000USD","Population","Arable Land/Acres","GDP Per Capita(US$)","Imports"]
```

Model Evaluation after 2nd iteration



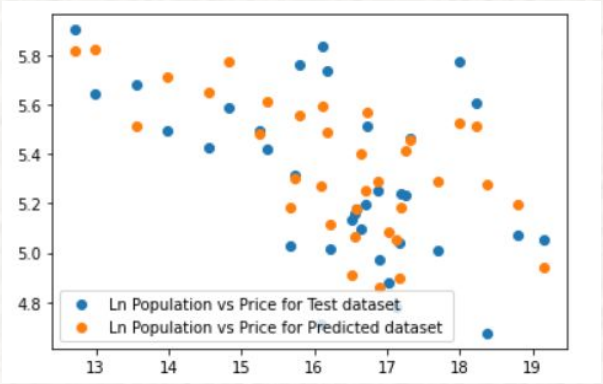


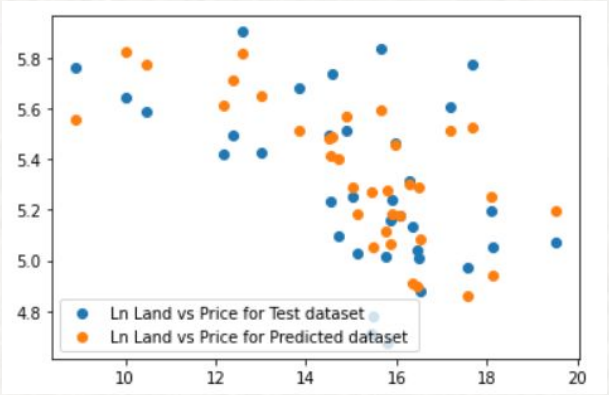
Model Evaluation after 2nd iteration

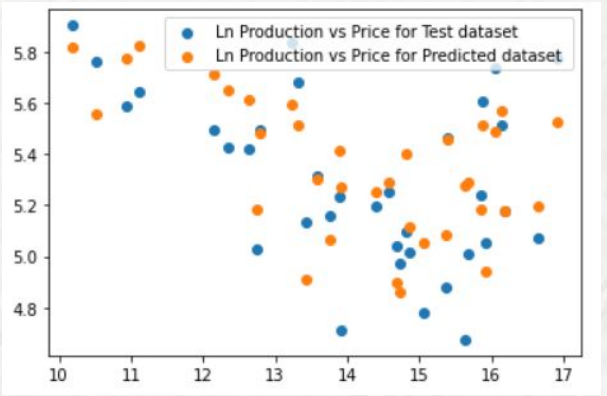
```
#calculating MSE:

def mean_squared_error(target, pred):
    n=target.shape[0]
    diff=np.sum((target-pred)**2)
    return diff/n
    pass


target1 = prepare_target(df_target_test1)
mse1=mean_squared_error(target1,pred1)

print("Mean Squared Error is - ",mse1)

def r2_score(y, yhat):
    R2=1 - np.sum((yhat - y)**2) / np.sum((y - np.mean(y))**2)
    return R2

#k -> number of independent variables
def adjusted_r2(y,p,R2):
    n_hello=y.shape[0]
    adj_rsquared = 1 - (1 - R2) * ((n_hello - 1)/(n_hello-p-1))
    return adj_rsquared

r2_1=r2_score(target1,pred1)

adjusted_r2_1=adjusted_r2(target1,5,r2_1)

print("Adjusted r2 - ",adjusted_r2_1)
```

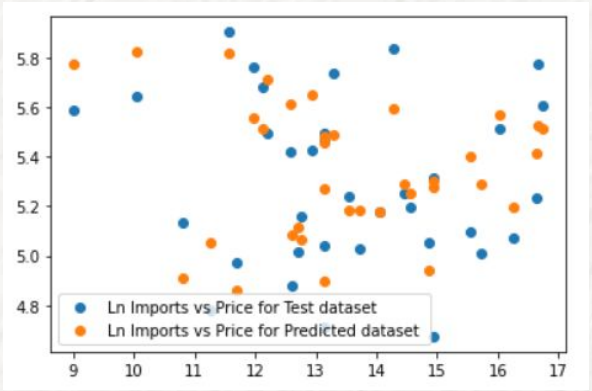
```
Mean Squared Error is -  2322.5786594476353
Adjusted r2 -  0.428889939738082
```

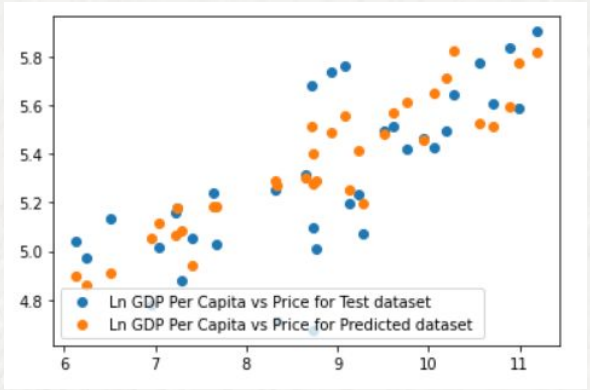
Model Evaluation after 3rd iteration











Model Evaluation after 3nd iteration

```
#calculating MSE:
def mean_squared_error(target, pred):
    n=target.shape[0]
    diff=np.sum((target-pred)**2)
    return diff/n

mse2=mean_squared_error(array_sum2,array_sum_pred2)

print("Mean Squared Error is - ",mse2)

#calculating adjusted r2 value:
def r2_score(y, yhat):
    R2=1 - np.sum((yhat - y)**2) / np.sum((y - np.mean(y))**2)
    return R2

#k -> number of independent variables
def adjusted_r2(y,p,R2):
    n=y.shape[0]
    adj_rsquared = 1 - (1 - R2) * ((n - 1)/(n-p-1))
    return adj_rsquared

r2_2=r2_score(array_sum2,array_sum_pred2)

adjusted_r2_2=adjusted_r2(array_sum2,5,r2_2)
print("Adjusted r2 - ",adjusted_r2_2)
```

```
Mean Squared Error is -  1889.5566035346424
Adjusted r2 -  0.5353684958464591
```

Overall,

Model is able to predict the Price index of fruits and vegetables in a country fairly accurately!

But!

Model is able to unable to take into account unaccounted factors such as government policies and climate factors.

Thank you!