DSC 680 Project 1 Global Food Waste Analysis

December 22, 2024

- Sauda Haywood
- DSC 680 Project 1

1 Data Exploration

[1]:
```python
# Import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import StandardScaler
```

[2]:
```python
# Uploaded CSV file
data = pd.read_csv('data.csv')
# Display the first few rows of the dataset
data.head()
```

[2]:
```
   m49_code  country region  cpc_code     commodity  year  loss_percentage  \
0       104  Myanmar    NaN  23161.02  Rice, milled  2015             1.78
1       104  Myanmar    NaN  23161.02  Rice, milled  2015            11.77
2       104  Myanmar    NaN  23161.02  Rice, milled  2015             5.88
3       104  Myanmar    NaN  23161.02  Rice, milled  2015             3.57
4       104  Myanmar    NaN  23161.02  Rice, milled  2015            17.65

  loss_percentage_original loss_quantity activity food_supply_stage  \
0                    1.78%      26.12kgs  Storage           Storage
1                   11.77%      88.18kgs  Storage           Storage
2                    5.88%      44.09kgs  Storage           Storage
3                    3.57%      52.24kgs  Storage           Storage
4                   17.65%     132.27kgs  Storage           Storage

                     treatment cause_of_loss sample_size  \
0  30 days storage, with trapping       Rodents         NaN
1    60 days storage, no trapping       Rodents         NaN
2    30 days storage, no trapping       Rodents         NaN
3  60 days storage, with trapping       Rodents         NaN
4    90 days storage, no trapping       Rodents         NaN
```

```
  method_data_collection                                         reference  \
0  Controlled Experiment  Dr Steven Belmain (2015), context post-harvest…
1  Controlled Experiment  Dr Steven Belmain (2015), context post-harvest…
2  Controlled Experiment  Dr Steven Belmain (2015), context post-harvest…
3  Controlled Experiment  Dr Steven Belmain (2015), context post-harvest…
4  Controlled Experiment  Dr Steven Belmain (2015), context post-harvest…

   url                                    notes
0  NaN   Reference has been generated automatically
1  NaN   Reference has been generated automatically
2  NaN   Reference has been generated automatically
3  NaN   Reference has been generated automatically
4  NaN   Reference has been generated automatically
```

The dataset includes the following variables:

- m49_code: A numerical code representing a country or region.
- country: The name of the country where the data was collected.
- region: The geographical region of the country.
- cpc_code: A code representing the commodity classification.
- commodity: The type of food commodity (e.g., milled rice).
- year: The year when the data was collected.
- loss_percentage: The percentage of food lost during the supply stage.
- loss_percentage_original: The original loss percentage before any transformations.
- loss_quantity: The quantity of food lost (e.g., in kilograms).
- activity: Specific activities related to the data collection or food supply stage.
- food_supply_stage: The stage in the food supply chain being analyzed (e.g., storage).
- treatment: The storage intervention applied (e.g., trapping, duration).
- cause_of_loss: Factors contributing to the food loss (e.g., rodents, poor storage methods).
- sample_size: The size of the sample used for data collection.
- method_data_collection: The methodology employed to collect the data.
- reference: References for the data source or study.
- url: A URL linking to additional resources or data.
- notes: Additional notes or comments about the data.

The target variable for the model will be loss_percentage

[3]:
```python
# Check the shape of the dataset
print(f"Dataset contains {data.shape[0]} rows and {data.shape[1]} columns.")
```

```
Dataset contains 25416 rows and 18 columns.
```

[4]:
```python
# Check basic info about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25416 entries, 0 to 25415
Data columns (total 18 columns):
```

```
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   m49_code                   25416 non-null  int64
 1   country                    25416 non-null  object
 2   region                     1214 non-null   object
 3   cpc_code                   25416 non-null  object
 4   commodity                  25416 non-null  object
 5   year                       25416 non-null  int64
 6   loss_percentage            25416 non-null  float64
 7   loss_percentage_original   25416 non-null  object
 8   loss_quantity              539 non-null    object
 9   activity                   22608 non-null  object
 10  food_supply_stage          22025 non-null  object
 11  treatment                  1320 non-null   object
 12  cause_of_loss              1002 non-null   object
 13  sample_size                1192 non-null   object
 14  method_data_collection     25061 non-null  object
 15  reference                  5113 non-null   object
 16  url                        22123 non-null  object
 17  notes                      2277 non-null   object
dtypes: float64(1), int64(2), object(15)
memory usage: 3.5+ MB
```

[5]:
```
# Check for missing values
missing_values = data.isnull().sum()
print("Missing values per column:")
print(missing_values[missing_values > 0])
```

```
Missing values per column:
region                    24202
loss_quantity             24877
activity                   2808
food_supply_stage          3391
treatment                 24096
cause_of_loss             24414
sample_size               24224
method_data_collection      355
reference                 20303
url                        3293
notes                     23139
dtype: int64
```

- Many empty rows due to data not being provided

[25]:
```
# Calculate mean loss percentage for each activity
activity_loss = (
    data.groupby('activity')['loss_percentage']
    .mean()
```

```
    .sort_values(ascending=False)
)


# Print the top 10
print(activity_loss.head(10))
```

```
activity
Harvesting, Packaging, Sorting, Storage
39.000000
Farm, Marketing, Storage, Transportation
38.000000
Milling, Storage
35.000000
Sorting, Washing
33.500000
Farm, Retailing, Trading, Wholesale
30.863636
Handling, Marketing, Storage
27.950000
Dewatering
25.150000
Bagging, Cleaning, Collection, Distribution, Drying, Field, Handling,
Harvesting, Sorting, Storage, Threshing    25.000000
Marketing, Retailing
25.000000
Consumption, Retailing
23.854167
Name: loss_percentage, dtype: float64
```

[6]:

```
# Calculate the mean loss percentage by commodity
commodity_loss = (
    data.groupby('commodity')['loss_percentage']
    .mean()
    .sort_values(ascending=False)
)

# Display the top 10 commodities by mean loss percentage
print(commodity_loss.head(10))
```

```
commodity
Snails, fresh, chilled, frozen, dried, salted or in brine, except sea snails
50.000000
Grapefruit juice
41.890548
Meat of pig with the bone, fresh or chilled
40.910000
Orange juice
```

```
40.300010
Pineapple juice
40.016635
Canned mushrooms
34.000000
Broad beans and horse beans, green
30.000000
Plantains and cooking bananas
28.272222
Uncooked pasta, not stuffed or otherwise prepared
28.000000
Other legumes, for forage
26.700000
Name: loss_percentage, dtype: float64
```

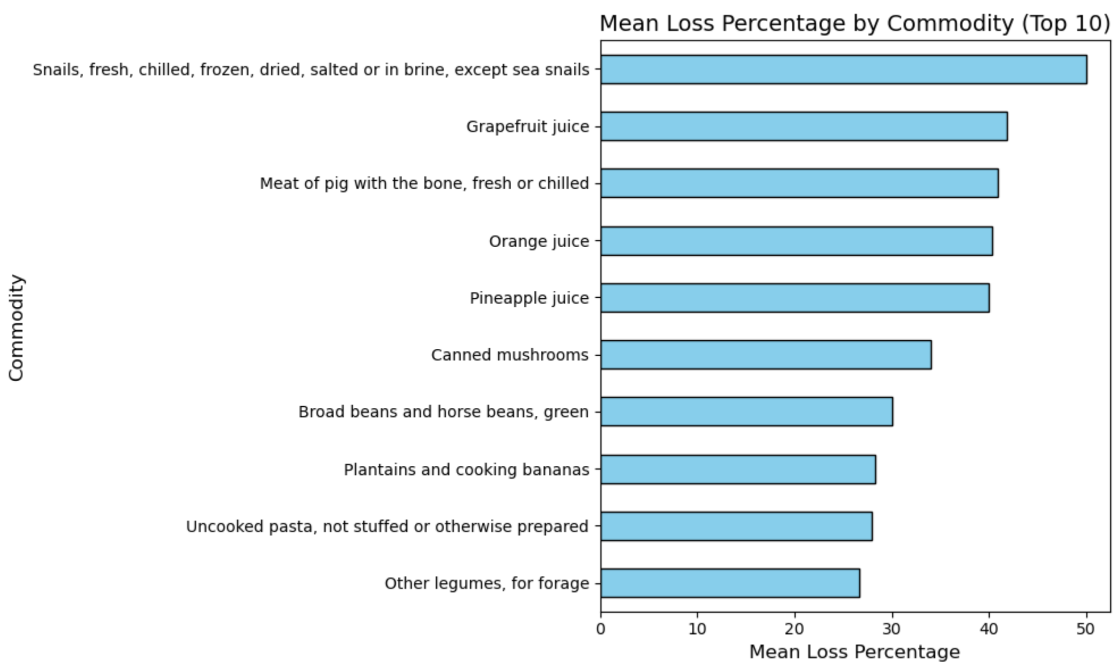
[12]:

```
# Calculate mean loss percentage for each cause of loss
cause_loss = (
    data.groupby('cause_of_loss')['loss_percentage']
    .mean()
    .sort_values(ascending=False)
)

# Print the top 10
print(cause_loss.head(10))
```

```
cause_of_loss
Losses In Marine Shipments
55.0
Measured In May; Due To Fruit Flies
50.0
Rejected because it does not meet European import criteria
50.0
Rejected Fruits Could Be Immature, Over Ripe, Bruised Or Fly Infested, With A
Low Chance Of Commercializationthey'Re Also Picked And Then Sorted For
Marketability, An Un Measured Percent Also Stays In The Fsc As Animal Feed
50.0
Literacy and technology exposure: 60 -90 % of producers did not pre-cool produce
after harvest.  Used vehicles or head loads to convey produce to market.
Producers had no storage facilities ; Marketers preferred the use of polythene
materials for packaging.    50.0
over-ripeness, rotting or excessive bruising, inappropriate postharvest
practices of middlemen
50.0
Trimming &collection
47.5
Storage cassava Chips
45.0
Higher Losses In July Due To Fruit Flies
```

```
43.5
Chemicals Repening Agents
43.0
Name: loss_percentage, dtype: float64
```

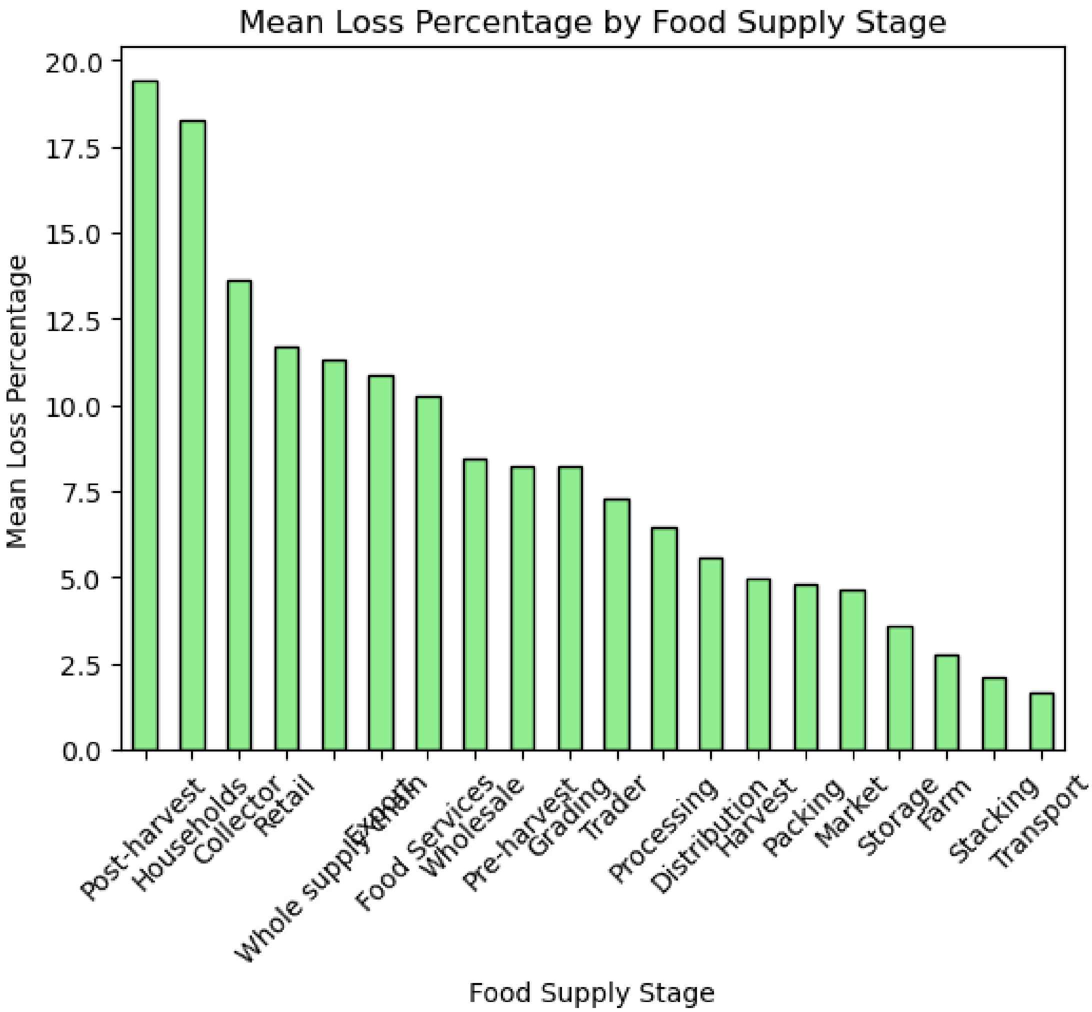
[7]:
```python
# Plot a horizontal bar chart for the top 10 commodities
commodity_loss.head(10).sort_values(ascending=True).plot(
    kind='barh',
    color='skyblue',
    edgecolor='black',
    figsize=(10, 6)
)
plt.title('Mean Loss Percentage by Commodity (Top 10)', fontsize=14)
plt.xlabel('Mean Loss Percentage', fontsize=12)
plt.ylabel('Commodity', fontsize=12)
plt.tight_layout()
plt.show()
```



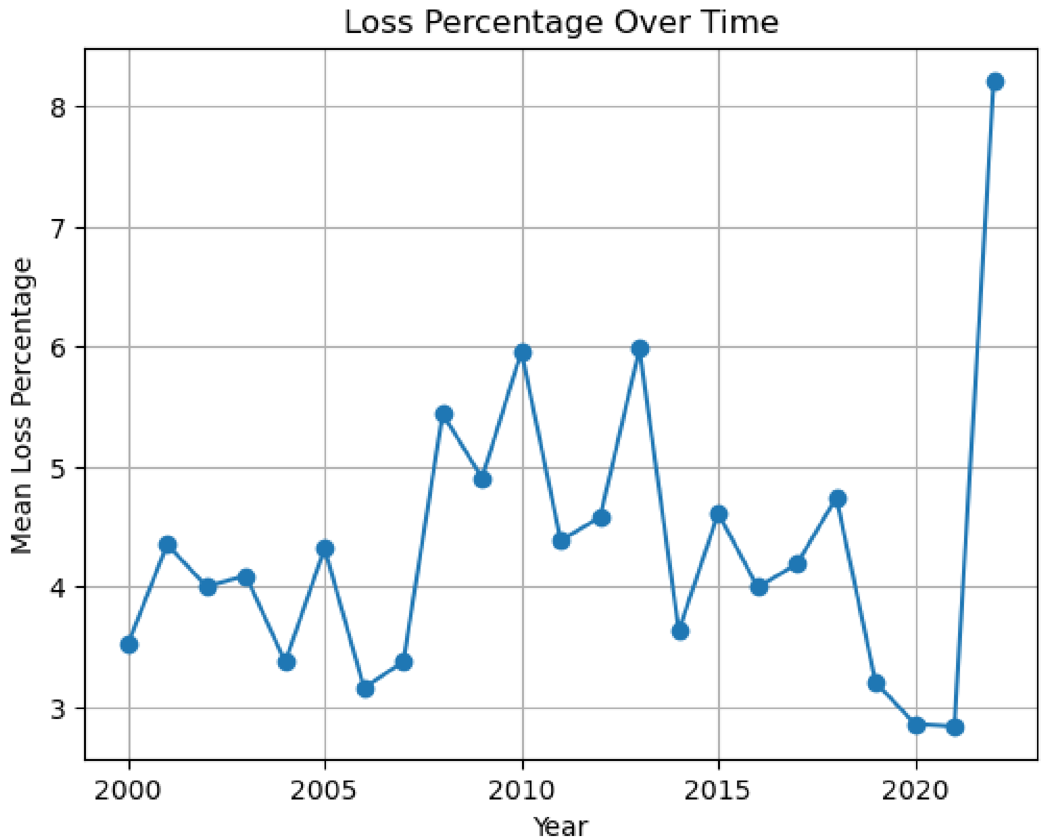
[11]:
```python
# Calculate mean loss percentage for each food supply stage
stage_loss = (
    data.groupby('food_supply_stage')['loss_percentage']
    .mean()
    .sort_values(ascending=False)
)
# Print the top 10
print(stage_loss.head(10))
```

```python
# Plot
stage_loss.plot(kind='bar', color='lightgreen', edgecolor='black')
plt.title('Mean Loss Percentage by Food Supply Stage')
plt.xlabel('Food Supply Stage')
plt.ylabel('Mean Loss Percentage')
plt.xticks(rotation=45)
plt.show()
```

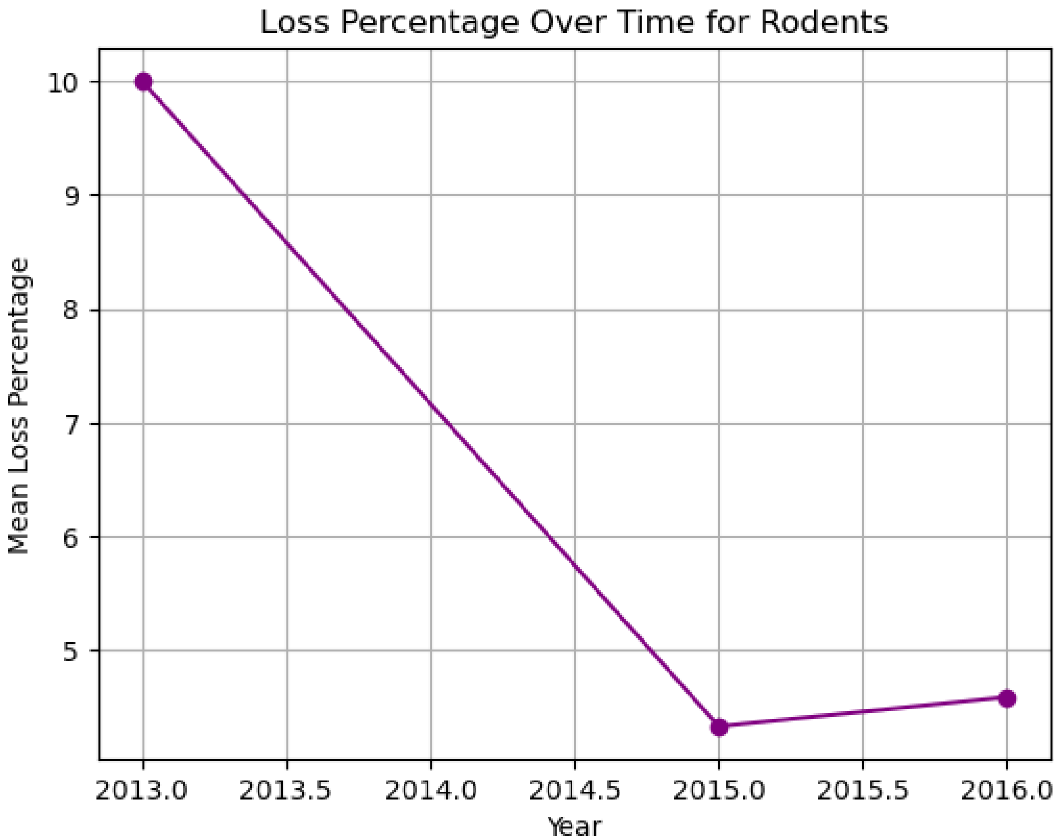
```
food_supply_stage
Post-harvest          19.413178
Households            18.267543
Collector             13.650000
Retail                11.685344
Whole supply chain    11.280271
Export                10.877800
Food Services         10.258000
Wholesale              8.413034
Pre-harvest            8.240909
Grading                8.227273
Name: loss_percentage, dtype: float64
```



[8]:
```python
mean_loss_by_year = data.groupby('year')['loss_percentage'].mean()

mean_loss_by_year.plot(kind='line', marker='o')
plt.title('Loss Percentage Over Time')
plt.xlabel('Year')
plt.ylabel('Mean Loss Percentage')
plt.grid(True)
plt.show()
```



[9]:
```
# Filter data for a specific cause of loss (e.g., "Rodents")
filtered_cause = data[data['cause_of_loss'] == 'Rodents']

# Group by year and calculate mean loss percentage
trend_by_year = (
    filtered_cause.groupby('year')['loss_percentage']
    .mean()
)

trend_by_year.plot(kind='line', marker='o', color='purple')
plt.title('Loss Percentage Over Time for Rodents')
plt.xlabel('Year')
plt.ylabel('Mean Loss Percentage')
plt.grid(True)
plt.show()
```



2 Model Building

2.1 Clean Data

[13]:
```python
# Check for missing values
print(data.isnull().sum())
```

```
m49_code                        0
country                         0
region                      24202
cpc_code                        0
commodity                       0
year                            0
loss_percentage                 0
loss_percentage_original        0
loss_quantity               24877
activity                     2808
food_supply_stage            3391
treatment                   24096
cause_of_loss               24414
```

```
sample_size                 24224
method_data_collection        355
reference                   20303
url                          3293
notes                       23139
dtype: int64
```

[14]:
```python
# Drop columns with excessive missing values
data = data.drop(columns=['region','method_data_collection',␣
 ↪'loss_percentage_original', 'treatment',
                          'sample_size', 'reference', 'url', 'loss_quantity',␣
 ↪'notes'])
```

[15]:
```python
# Handling missing values
data = data.dropna(subset=['loss_percentage'])  # Drop rows with missing target
data.fillna({'cause_of_loss': 'Unknown'}, inplace=True)
```

[16]:
```python
# One-hot encode categorical variables
data_encoded = pd.get_dummies(data, columns=['commodity', 'food_supply_stage'],␣
 ↪drop_first=True)
```

[17]:
```python
# One-hot encode 'cpc_code'
data = pd.get_dummies(data, columns=['cpc_code'], drop_first=True)
```

[18]:
```python
# One-hot encode 'cause_of_loss'
data = pd.get_dummies(data, columns=['cause_of_loss'], drop_first=True)
```

[19]:
```python
# Check for non-numeric columns
non_numeric_columns = data.select_dtypes(include=['object']).columns
print("Remaining non-numeric columns:", non_numeric_columns)
```

```
Remaining non-numeric columns: Index(['country', 'commodity', 'activity',
'food_supply_stage'], dtype='object')
```

[20]:
```python
# One-hot encode the remaining non-numeric columns
data_encoded = pd.get_dummies(data, columns=['country', 'commodity',␣
 ↪'activity', 'food_supply_stage'], drop_first=True)
```

2.2 Model 1 Linear Regression Model

2.2.1 Splitting the Data

[21]:
```python
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = data_encoded.drop(columns=['loss_percentage'])
y = data_encoded['loss_percentage']

# Split the data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

2.2.2 Train the Linear Regression Model

[22]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and fit the model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on test set
y_pred = lr_model.predict(X_test)
```

2.2.3 Make Predictions

[23]:
```
# Make predictions on the test set
y_pred = lr_model.predict(X_test)
```

2.2.4 Evaluate the model

[24]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Calculate evaluation metrics
rmse = mean_squared_error(y_test, y_pred, squared=False)  # Root Mean Squared␣
↪Error
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
r2 = r2_score(y_test, y_pred)  # R-squared (coefficient of determination)

# Print metrics
print(f'Linear Regression RMSE: {rmse}')
print(f'Linear Regression MAE: {mae}')
print(f'Linear Regression R^2: {r2}')
```

```
Linear Regression RMSE: 16396338.77815438
Linear Regression MAE: 361902.9301619577
Linear Regression R^2: -9178495110209.982
```

- RMSE (Root Mean Squared Error): 16,396,338.78 , This extremely high value indicates that the predictions are far off from the actual values.
- MAE (Mean Absolute Error): 361,902.93, very large number. poor fit of the model.
- R^2 (R-Squared): -9,178,495,110,209.982 , very large number. This model is not suitable.

2.3 Model 2: Random Forest

[26]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[27]:
```
# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)
```

[27]:
```
RandomForestRegressor(random_state=42)
```

[28]:
```
# Predict on the test set
y_pred = rf_model.predict(X_test)
```

[29]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Calculate evaluation metrics
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f'Random Forest RMSE: {rmse}')
print(f'Random Forest R^2: {r2}')
```

```
Random Forest RMSE: 2.9977699346269353
Random Forest R^2: 0.6931866065914762
```

- RMSE (Root Mean Squared Error): 2.997 , This lower RMSE values indicate better performance.
- R^2 (R-Squared): 0.693, This indicates that 69.3% of the variance in loss_percentage is explained by the model.

[ ]:
```
Conclusion: Random forest is better option model for this data.
```