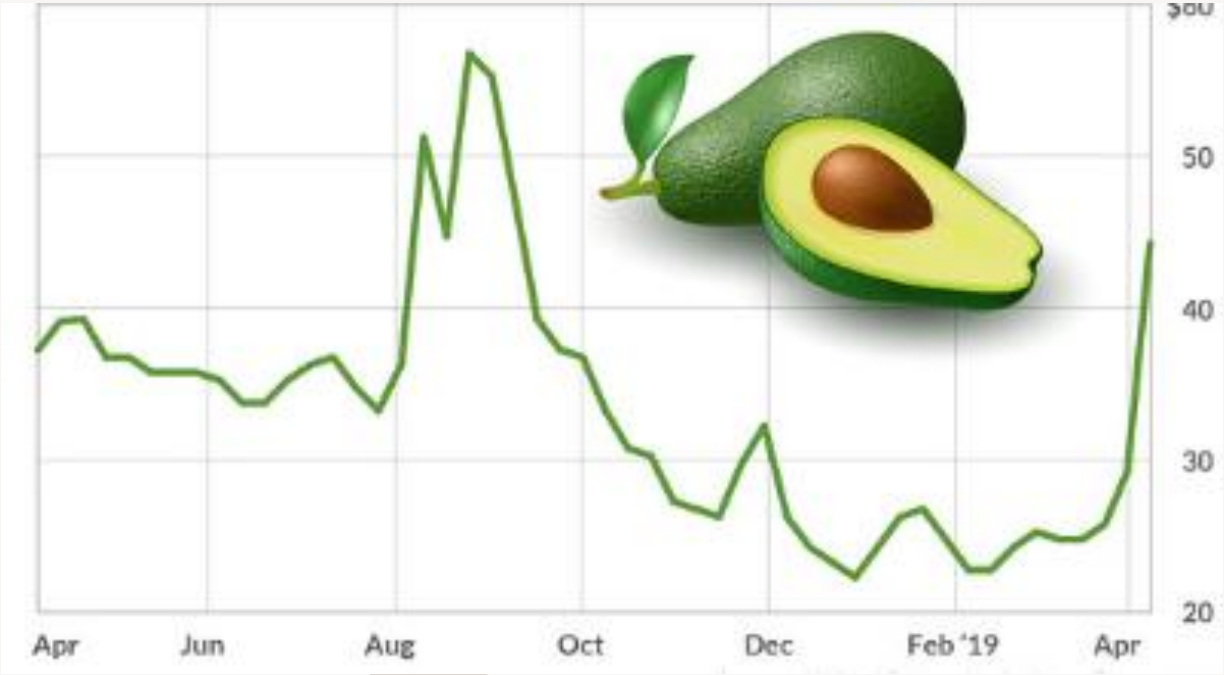
Predicting Avocado Prices Using Machine Learning

Tasmita Tanjim

Student ID: 19723



Project Overview

Overview

- Avocado prices fluctuate across seasons and regions, impacting retailers and suppliers..
- This project uses machine learning to predict avocado prices based on sales volume, packaging types, region, and year..

Problem

- Traditional pricing estimates may overlook complex patterns in the data.
- Manual methods can be inconsistent and lack scalability

Solution

- Trained ML models (Decision Tree, Random Forest, XGBoost) to predict average prices.
- Used preprocessing pipelines and hyperparameter tuning for accurate regression modeling.

Dataset Description

- Source: Kaggle – Avocado Prices Dataset
- Size: ~18,000 records from 2015–2018
- Data Shape: (18249, 13)
- Target Variable: AveragePrice
- Feature used:

  Sales volumes by size(Small, Large, Xlarge)

  Bag types

  Type(organic/conventional). Region, year
- No missing values in the dataset
- Problem Type: Regression

```
df = df.rename(columns={
    "4046": "SmallAvocados",
    "4225": "LargeAvocados",
    "4770": "XLAvocados",
    "Total Volume": "TotalVolume",
    "Total Bags": "TotalBags",
    "Small Bags": "SmallBags",
    "Large Bags": "LargeBags",
    "XLarge Bags": "XLargeBags"
})
```

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

✓ 0.0s

```
Date            0
AveragePrice    0
TotalVolume     0
SmallAvocados   0
LargeAvocados   0
XLAvocados      0
TotalBags       0
SmallBags       0
LargeBags       0
XLargeBags      0
type            0
year            0
region          0
dtype: int64
```

```
df.describe()
```

✓ 0.0s

	AveragePrice	TotalVolume	SmallAvocados	LargeAvocados	XLAvocados	TotalBags	SmallBags	LargeBags	XLargeBags	year
count	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	18249.000000	18249.000000
mean	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974e+04	2.396392e+05	1.821947e+05	5.433809e+04	3106.426507	2016.147899
std	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05	9.862424e+05	7.461785e+05	2.439660e+05	17692.894652	0.939938

Project Workflow

1. Data Collection

→ Collected the avocado dataset from Kaggle

2. Data Preprocessing

→ Applied StandardScaler to numeric features

→ One-hot encoded using ColumnTransformer

3. EDA & Model training

→ Trained 5 regression models: Ridge, Lasso, DT, RF, XGBoost

→Evaluated models on test data using MAE, RMSE, and R^2

4. Hyperparameter Tuning

→ Used GridSearchCV with 5-fold for DT, RF & XGBoost

→ Selected the best-performing parameter sets

5. Final Model Evaluation

→ Re-trained models on the full training set using the best parameters

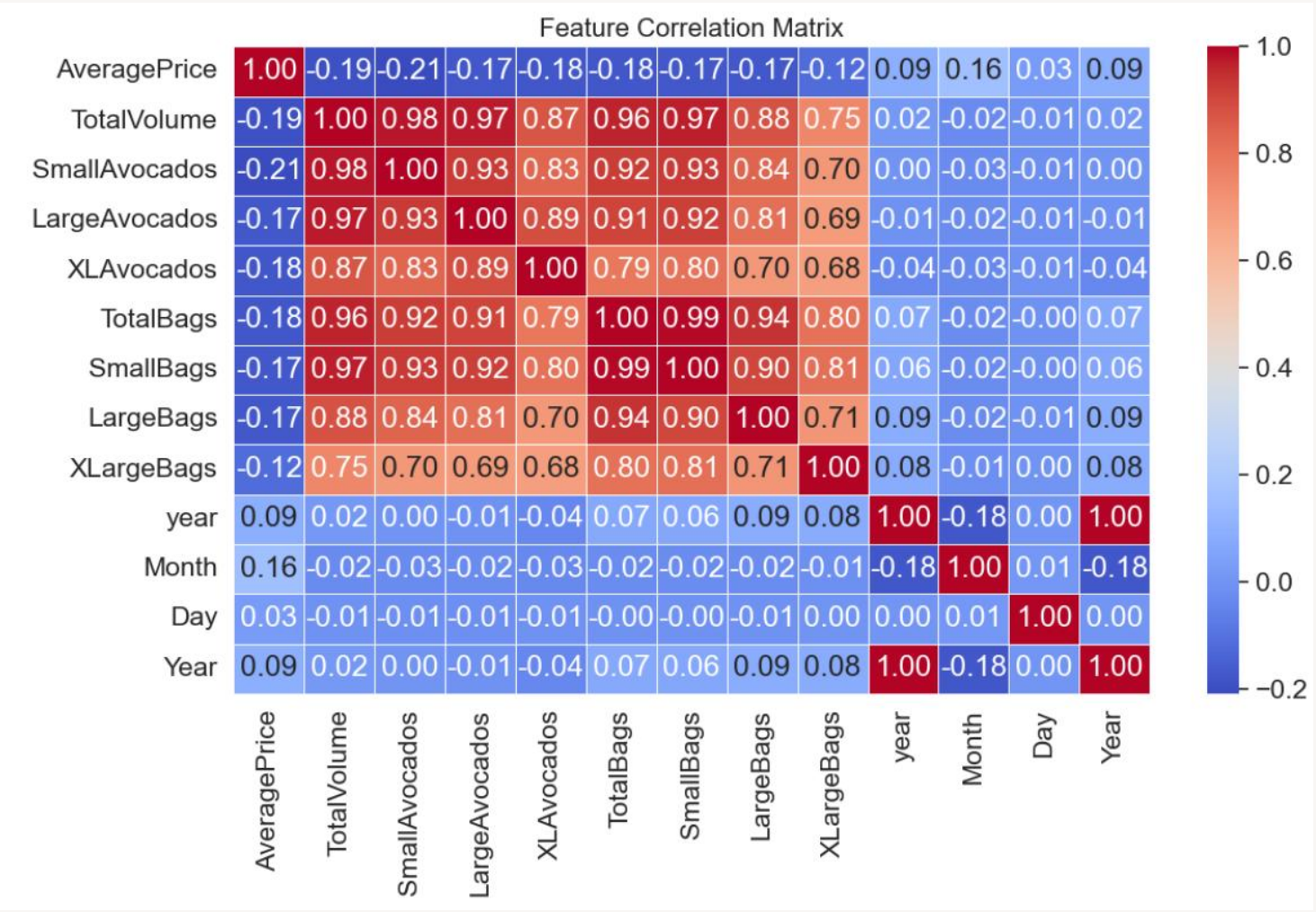
→Evaluated all tuned models on test data

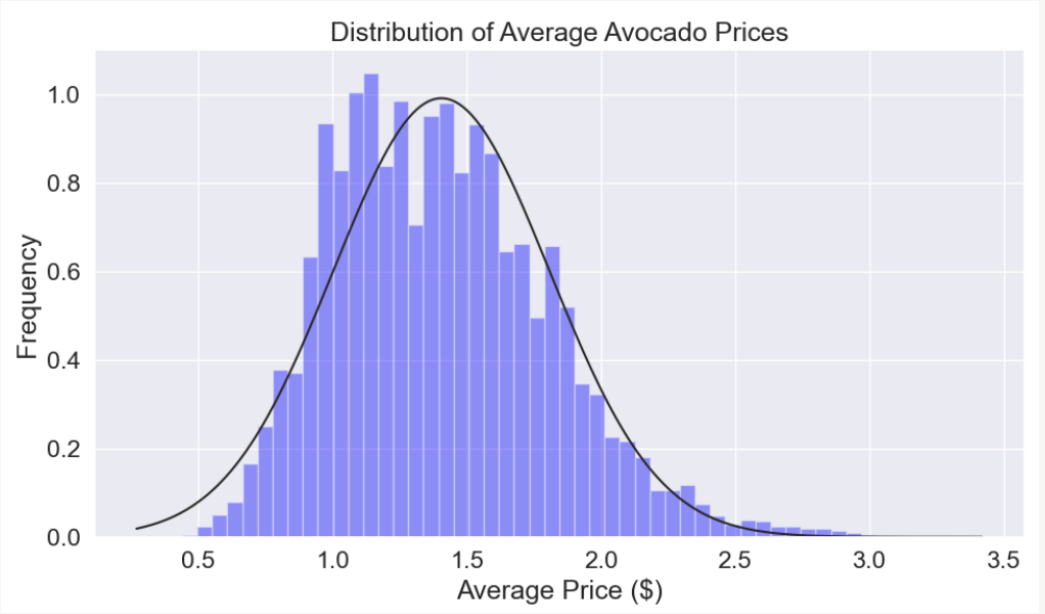
6. Result Interpretation

→ Visualized actual vs predicted scatter plots

→ Showed a performance comparison table

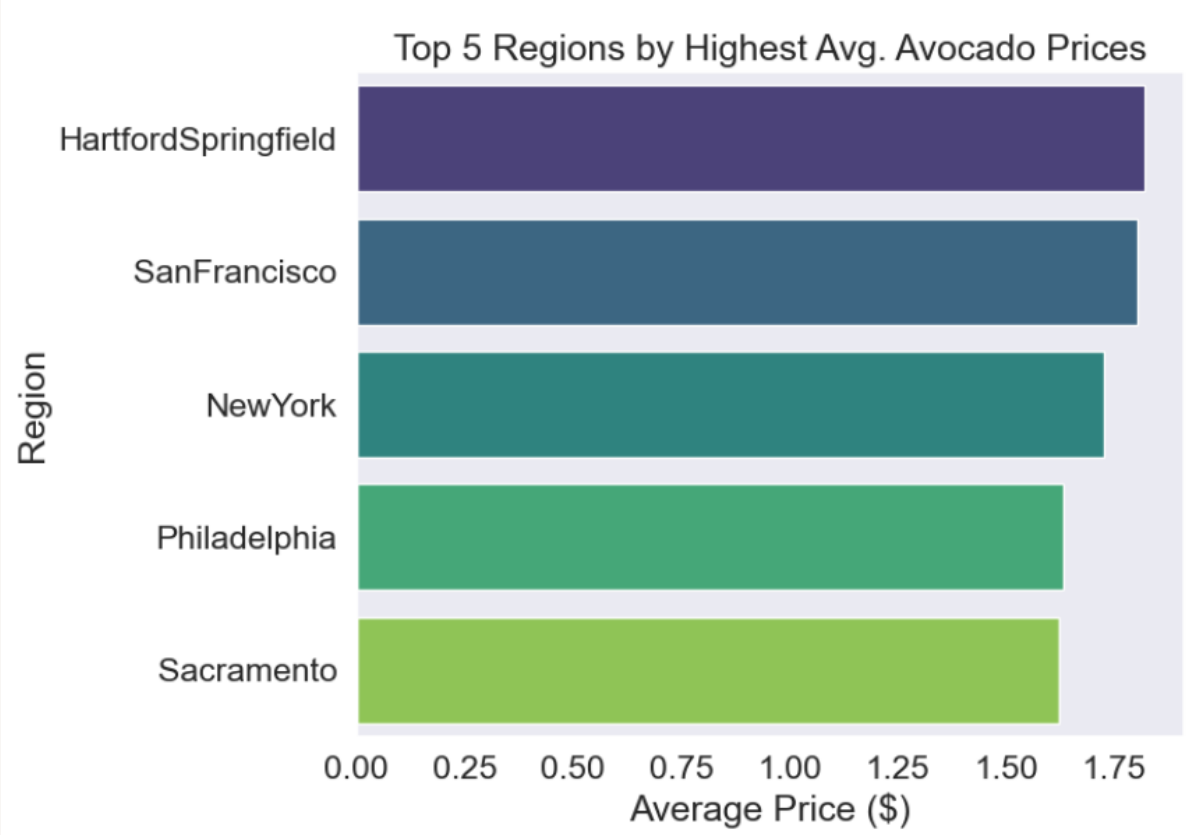
Data Visualization

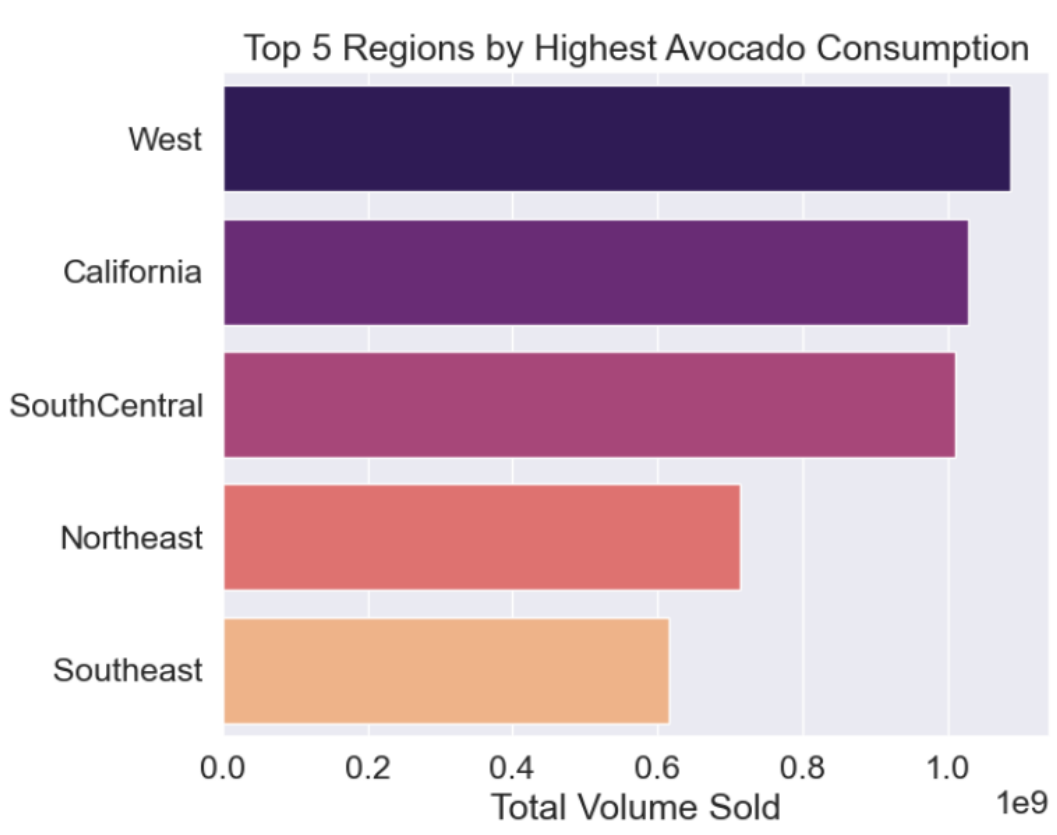




- Most avocado prices are between $1.0 and $1.5.
- Avocados are generally affordable, with high prices being less common

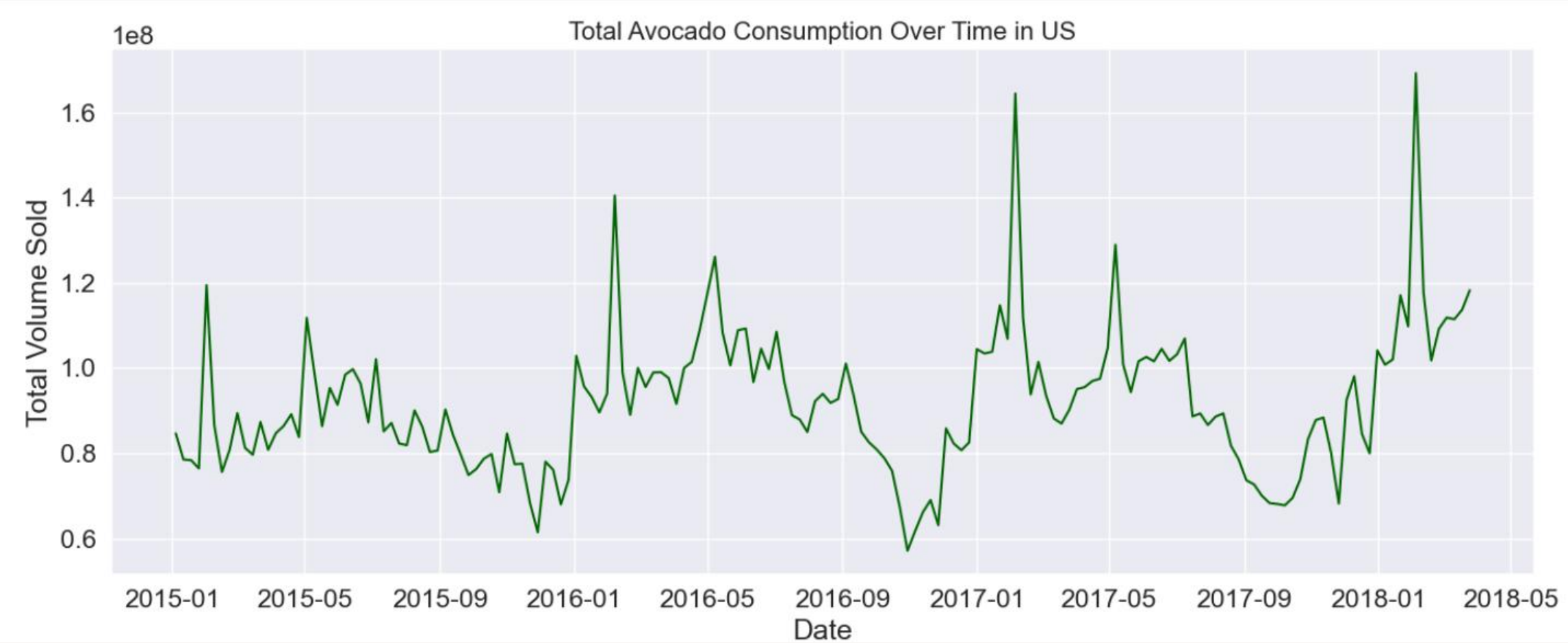
Avocado Prices and Consumption – Top 5 Regions





- **Highest Prices:** Cities like HartfordSpringfield & San Francisco have the highest avocado prices..
- **Highest Consumption**: Regions like the West and California buy the most avocados.
- High prices don’t always mean high consumption. Some areas pay more, others buy more.

Time trend



- Avocado sales show **seasonal spikes**, especially early each year.
- Highest consumption peaks occur around **January and February**, possibly due to various events.

Feature Scaling (StandardScaler)

```
print("Before scaling (Numeric Features):")
display(X[numeric_features].describe().T[["mean", "std", "min", "max"]])
```

✓ 0.0s

Before scaling (Numeric Features):

	mean	std	min	max
TotalVolume	850644.013009	3.453545e+06	84.56	62505646.52
SmallAvocados	293008.424531	1.264989e+06	0.00	22743616.17
LargeAvocados	295154.568356	1.204120e+06	0.00	20470572.61
XLAvocados	22839.735993	1.074641e+05	0.00	2546439.11
TotalBags	239639.202060	9.862424e+05	0.00	19373134.37
SmallBags	182194.686696	7.461785e+05	0.00	13384586.80
LargeBags	54338.088145	2.439660e+05	0.00	5719096.61
XLargeBags	3106.426507	1.769289e+04	0.00	551693.65
year	2016.147899	9.399385e-01	2015.00	2018.00

```
scaler = StandardScaler()
scaled_numeric = scaler.fit_transform(X[numeric_features])
scaled_df = pd.DataFrame(scaled_numeric, columns=numeric_features)

print("After scaling(Numeric Features):")
display(scaled_df.describe().T[["mean", "std", "min", "max"]])
```

✓ 0.0s

After scaling(Numeric Features):

	mean	std	min	max
TotalVolume	-2.491903e-17	1.000027	-0.246293	17.853158
SmallAvocados	-6.229757e-18	1.000027	-0.231636	17.748155
LargeAvocados	-2.180415e-17	1.000027	-0.245127	16.755775
XLAvocados	5.295294e-17	1.000027	-0.212540	23.483836
TotalBags	3.737854e-17	1.000027	-0.242989	19.400930
SmallBags	-9.344636e-18	1.000027	-0.244177	17.693827
LargeBags	-1.868927e-17	1.000027	-0.222734	23.220099
XLargeBags	2.803391e-17	1.000027	-0.175580	31.006925
year	-1.021182e-13	1.000027	-1.221282	1.970504

Data Preprocessing (onehotencoder)

```
encoder = OneHotEncoder(sparse_output=False, handle_unknown="ignore")
encoded_cat = encoder.fit_transform(X[categorical_features])
encoded_cat_df = pd.DataFrame(encoded_cat, columns=encoder.get_feature_

print("\nAfter encoding (categorical features):")
print(f"Total encoded columns: {encoded_cat_df.shape[1]}")
print("Sample encoded column names:", list(encoded_cat_df.columns[:5]))
```

✓ 0.0s

```
After encoding (categorical features):
Total encoded columns: 56
Sample encoded column names: ['type_conventional', 'type_organic', 'region_
0        conventional
1        conventional
2        conventional
3        conventional
4        conventional
            ...
18244         organic
18245         organic
18246         organic
18247         organic
18248         organic
Name: type, Length: 18249, dtype: object
```

```
print("Before encoding (categorical features):")
for col in categorical_features:
    print(f"{col}: {X[col].nunique()} unique values")
```

✓ 0.0s

```
Before encoding (categorical features):
type: 2 unique values
region: 54 unique values
```

```
numeric_transformer = Pipeline(steps=[("scaler", StandardScaler())])

categorical_transformer = Pipeline(steps=[("encoder", OneHotEncoder(handle_

# Combining into a ColumnTransformer
preprocessor = ColumnTransformer(transformers=[
    ("num", numeric_transformer, numeric_features),
    ("cat", categorical_transformer, categorical_features)
])
```

✓ 0.0s

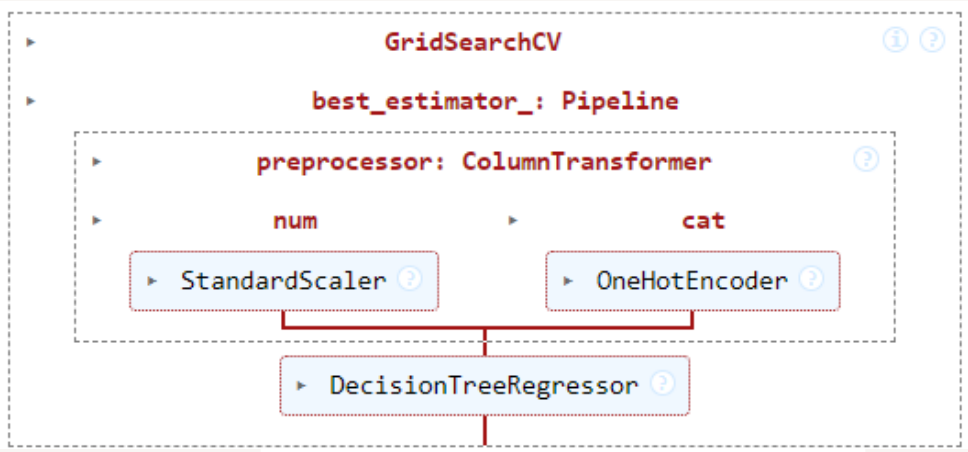
```python
baseline_models = {
    "Ridge": Ridge(),
    "Lasso": Lasso(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=50, random_stat
    "XGBoost": XGBRegressor(n_estimators=50, verbosity=0, random_state=
}

# Re-split train/test data to ensure consistency
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Store baseline results
baseline_results = []

# Evaluate each model without CV
for name, model in baseline_models.items():
    pipeline = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("regressor", model)
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    baseline_results.append({
        "Model": name,
        "MAE": round(mae, 4),
        "RMSE": round(rmse, 4),
        "R² Score": round(r2, 4)
    })
```

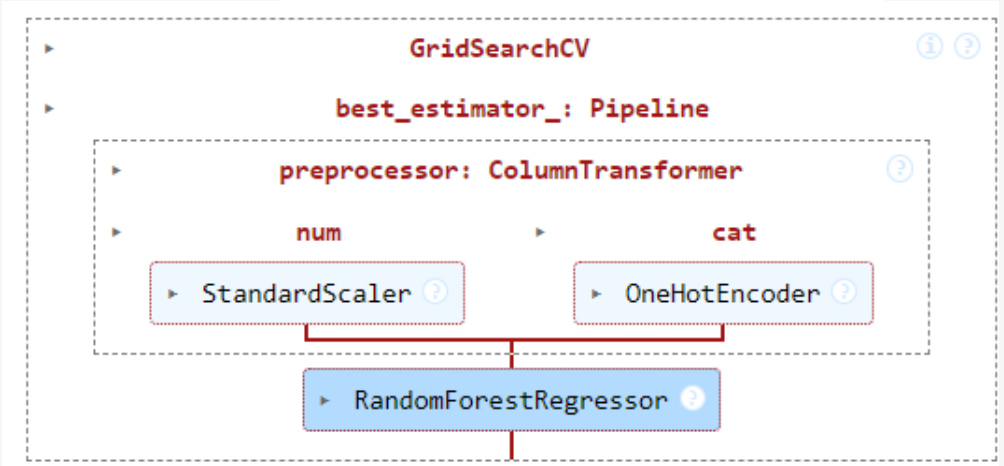
Model Selection & Training

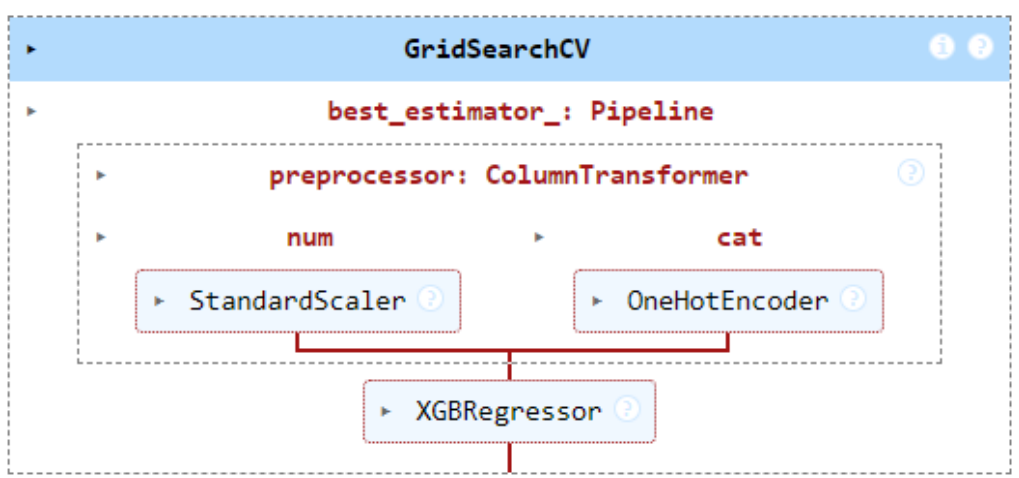
	Model	MAE	RMSE	R^2 Score
0	Random Forest	0.1077	0.1533	0.8537
1	XGBoost	0.1268	0.1726	0.8145
2	Decision Tree	0.1421	0.2182	0.7036
3	Ridge	0.2034	0.2701	0.5459
4	Lasso	0.3225	0.4009	-0.0002

- Random Forest got the best performance with low RMSE and high R^2
- **XGBoost** followed closely behind, showing strong predictive capability.
- **Lasso** performed poorly, with a near-zero R^2, indicating underfitting

Hyperparameter Tuning



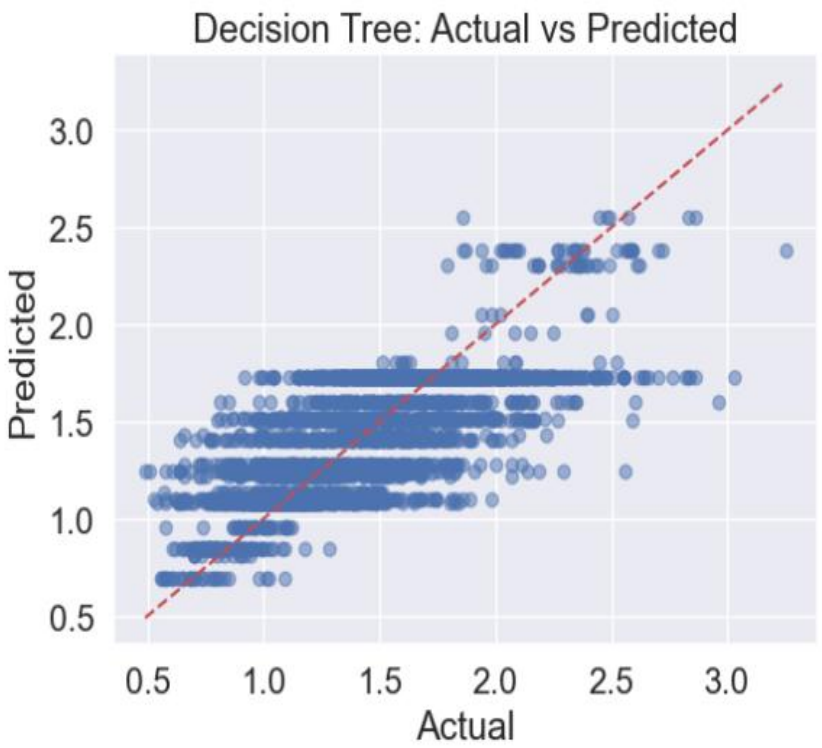


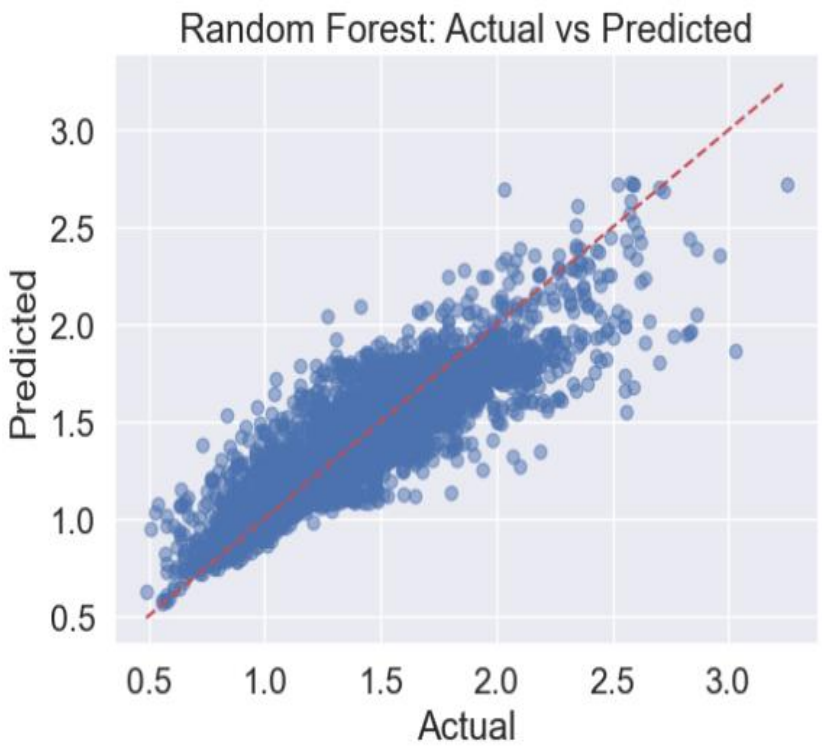


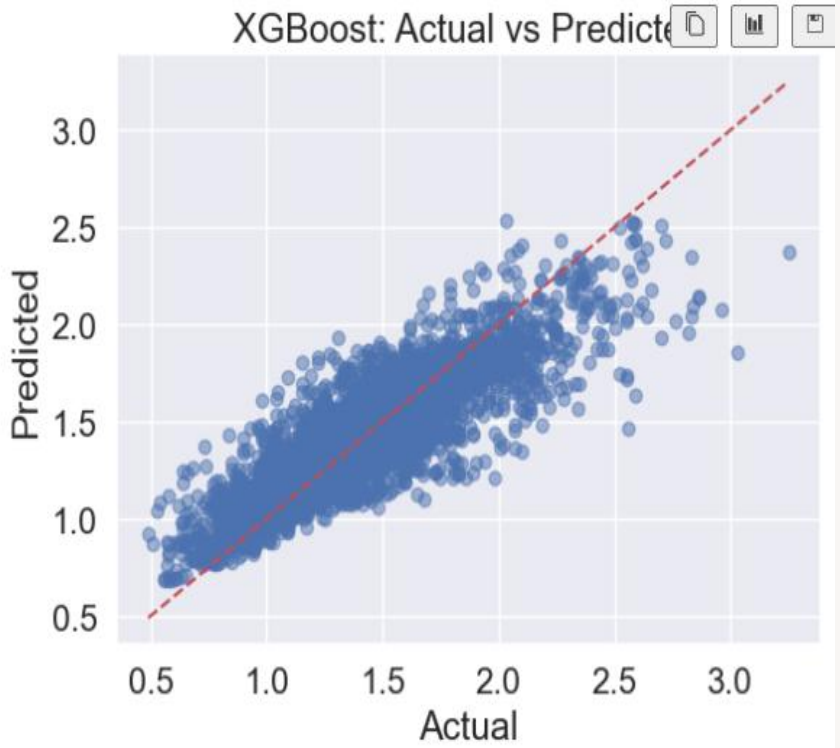

```
Model Evaluation Summary (After Hyperparameter Tuning)
Decision Tree
- Best Parameters: {'regressor__max_depth': 5, 'regres
- Best Cross-Validated RMSE: 0.2994

Random Forest
- Best Parameters: {'regressor__max_depth': 10, 'regre
- Best Cross-Validated RMSE: 0.281

XGBoost
- Best Parameters: {'regressor__learning_rate': 0.1, '
- Best Cross-Validated RMSE: 0.2579
```

XGBoost performed best with the lowest RMSE (0.2579).
Random Forest also improved (RMSE 0.281).

Conclusion







Model Name	R^2 Score
Random Forest	0.8537 (highest)
XGBoost	0.8145
Decision Tree	0.7036

Thank you!