Configuration Manual

MSc Research Project
Data Analytics

Ashok Saravanan Sundarrajan
Student ID: x21204764

School of Computing
National College of Ireland

Supervisor: Prashanth Nayak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ashok Saravanan Sundarrajan
Student ID:	x21204764
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prashanth Nayak
Submission Due Date:	14/07/2023
Project Title:	Configuration Manual
Word Count:	657
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ashok Saravanan Sundarrajan
Date:	18th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ashok Saravanan Sundarrajan
x21204764

1 Introduction

The rising demand for food, especially in developing countries, has necessitated the advancement of agricultural techniques. Accurate wheat yield prediction has emerged as a vital requirement to ensure food security and economic stability. This manual provides guidelines on setting up the system to run an ensemble machine learning approach for wheat yield prediction in India. The approach combines the strengths of Random Forest, Support Vector Machine, and Decision Tree models.

2 System Specification

The System specification for this research work includes the following machine con figuration.

2.1 Hardware Specifications

- Processor: 2.3 GHz Dual-Core Intel Core i5.
- RAM Memory: 8 GB 2133 MHz LPDDR3.
- Storage: 256GB SSD.
- Graphics: Intel Iris Plus Graphics 640 1536 MB.
- Operating System: Mac OS Ventura 13.4.1

2.2 Software Specifications

IDE	Google Colab, Jupyter Notebook
Programming Language	Python v3.9.1
Modules	Matplotlib, Pandas, Numpy, Scikit-learn
Computation	GPU
Number of GPU	1
GPU Type	Tesla K80 GPU-12GB

Table 1: Software Specification

3 Importing Required Libraries

This research includes, importing of libraries in the colab which uses python environment. the major advantage of using colab is that the python modules like pandas, numpy etc are preloaded into it.

the Figure 1 shows the Libraries and Dependencies that needs to be imported into this project.

Importing Required Libraries and Read Dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import preprocessing
from math import sqrt
from sklearn.model_selection import train_test_split, KFold, TimeSeriesSplit, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import VotingRegressor, StackingRegressor
```

Figure 1: Libraries and Dependencies

4 Loading Data

The data has been uploaded to the colab inbuilt storage and the data has been loaded into data frame using pandas for further processing [1]. Fig 2 shows the data frame after data loading into colab for data pre-processing.

```
[ ] df = pd.read_csv('data.csv')
```

```
df
```

	State	District	Crop	Crop_Year	Season	Area	Production	Yield
0	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Kharif	2439.6	3415.0	1.40
1	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Rabi	1626.4	2277.0	1.40
2	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Autumn	4147.0	3060.0	0.74
3	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Summer	4147.0	2660.0	0.64
4	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Autumn	4153.0	3120.0	0.75
...	...	...	...	...	...	...	...	...
345331	West Bengal	PURULIA	Wheat	2015	Rabi	855.0	1241.0	1.45
345332	West Bengal	PURULIA	Wheat	2016	Rabi	1366.0	2415.0	1.77
345333	West Bengal	PURULIA	Wheat	2017	Rabi	1052.0	2145.0	2.04
345334	West Bengal	PURULIA	Wheat	2018	Rabi	833.0	2114.0	2.54
345335	West Bengal	PURULIA	Wheat	2019	Rabi	516.0	931.0	1.80

345336 rows × 8 columns

Column names have white spaces so we can trim it.

Figure 2: Data Loading

[1] https://data.world/thatzprem/agriculture-india

5 Data Pre-Processing and Data Cleaning

In Data Pre-processing the column names were not properly formatted so we format using the strip method in python and then we check for NA values present in the dataset and it includes 4948 NA values in the Production Column. it is shown in Figure 3.

```
[ ] df.columns = df.columns.str.strip()
    print(df.columns)

    Index(['State', 'District', 'Crop', 'Crop_Year', 'Season', 'Area',
           'Production', 'Yield'],
          dtype='object')
```

Figure 3: Data Column Trimming

finally NA values are cleaned which is shown in in Figure 4.

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

State            0
District         0
Crop             9
Crop_Year        0
Season           0
Area             0
Production    4948
Yield            0
dtype: int64
```

there are 4948 na values in 4948 which are not required. so lets drop it.

```
[ ] df = df.dropna()
```

Figure 4: Dropping NA Values

we consider only Wheat for our analysis and we filter out it from different crops. the Data frame for wheat consist of 11208 rows and 8 columns.

```
df_wheat
```

	State	District	Crop	Crop_Year	Season	Area	Production	Yield
16914	Andhra Pradesh	ADILABAD	Wheat	1997	Rabi	3600.0	2000.0	0.56
204766	Meghalaya	WEST GARO HILLS	Wheat	1997	Rabi	4215.0	6811.0	1.62
204726	Meghalaya	EAST GARO HILLS	Wheat	1997	Rabi	72.0	83.0	1.15
39246	Assam	DIMA HASAO	Wheat	1997	Rabi	56.0	73.0	1.30
197366	Maharashtra	YAVATMAL	Wheat	1997	Rabi	18600.0	8700.0	0.47
...	...	...	...	...	...	...	...	...
332550	Uttarakhand	CHAMPAWAT	Wheat	2020	Rabi	4657.0	7358.0	1.58
332718	Uttarakhand	UDAM SINGH NAGAR	Wheat	2020	Rabi	105961.0	471000.0	4.45
332571	Uttarakhand	DEHRADUN	Wheat	2020	Rabi	14271.0	42482.0	2.98
332739	Uttarakhand	UTTAR KASHI	Wheat	2020	Rabi	9082.0	15203.0	1.67
332508	Uttarakhand	BAGESHWAR	Wheat	2020	Rabi	13858.0	23715.0	1.71

11208 rows × 8 columns

Figure 5: Data Frame for Wheat

We have Detected outliers using IQR method and its represented in the following Figure 6

```python
# Select numerical columns
num_cols = ["Area", "Production", "Yield"]

sns.set_style("whitegrid")

fig, axs = plt.subplots(ncols=3, figsize=(20, 8))

for i, col in enumerate(num_cols):
    sns.boxplot(y=df_wheat[col], ax=axs[i])
    axs[i].set_title('{}'.format(col), fontsize=15)
    axs[i].set_xlabel('')

plt.tight_layout()
plt.show()
```

Figure 6: Detecting Outliers in Data

6 Exploratory Data Analysis

Exploratory Data Analysis was implemented in this to understand the data better. the various plots and visualizations give us better insights about the crop varieties which helps us to build the model better. we have visualized several plots which has been shown in the following Figure 7

```python
df_wheat['Yield'].hist(bins=40)
plt.xlabel('Yield')
plt.ylabel('Frequency')
plt.title('Histogram of Yield')
plt.show()

plt.figure(figsize=(8,4))
sns.boxplot(x='Crop', y='Yield', data=df_wheat)
plt.xticks(rotation=90)
plt.title('Boxplot of Yield per Crop')
plt.show()

plt.figure(figsize=(8,4))
plt.scatter(df_wheat['Area'], df_wheat['Yield'])
plt.xlabel('Area')
plt.ylabel('Yield')
plt.title('Scatter plot of Area vs Yield')
plt.show()

sns.set_palette('PRGn')
total_area_per_crop = df.groupby('Crop')['Area'].sum().sort_values(ascending=False).head(20)
total_area_per_crop.plot(kind='bar', figsize=(8,6))
plt.ylabel('Total Area')
plt.title('Total Area per Crop')
plt.show()


crop_counts = df['Crop'].value_counts()
N = 10
colors = cm.viridis(np.linspace(0, 1, N+1))  # Creating a colormap using viridis
top_crops = crop_counts[:N]
top_crops['Other'] = crop_counts[N:].sum()
top_crops.plot(kind='pie', figsize=(8,6), autopct='%1.1f%%', colors=colors)
plt.title(f'Pie Chart of Top {N} Crop Frequencies')
plt.show()
```

```python
wheat_data = df[df['Crop']=='Wheat']
wheat_data.groupby('Crop_Year')['Yield'].mean().plot()
plt.ylabel('Yield')
plt.title('Time Series Plot of Yield for Wheat')
plt.show()

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Heatmap of Correlations')
plt.show()

total_production_per_state = df.groupby('State')['Production'].sum().sort_values(ascending=False).head(10)
total_production_per_state.plot(kind='bar', figsize=(8,4))
plt.ylabel('Total Production')
plt.title('Total Production per State')
plt.show()

crop = 'Wheat'
df_crop = df[df['Crop'] == crop]
production_trend = df_crop.groupby('Crop_Year')['Production'].sum()

plt.figure(figsize=(8,6))
plt.plot(production_trend.index, production_trend.values, marker='o')
plt.title(f'{crop} Production Over Time')
plt.xlabel('Year')
plt.ylabel('Production')
plt.grid(True)
plt.show()

plt.figure(figsize=(8,6))
sns.scatterplot(x='Area', y='Production', data=df_wheat.sample(1000))  # Using a sample for better visualization
plt.title('Area vs Production')
plt.show()
```

Figure 7: Exploratory Data Analysis

7 Model Building

7.1 Label Encoding

In this research, the dataset encompass categorical features such as the state, district, Crop type and Season where the wheat is grown. These categorisations, being non-numeric, need to be transformed into a machine-readable format. this is shown in figure 8.

```python
le = LabelEncoder()
```

```python
df_wheat_cleaned['State'] = le.fit_transform(df_wheat_cleaned['State'])
df_wheat_cleaned['District'] = le.fit_transform(df_wheat_cleaned['District'])
df_wheat_cleaned['Season'] = le.fit_transform(df_wheat_cleaned['Season'])
df_wheat_cleaned['Crop'] = le.fit_transform(df_wheat_cleaned['Crop'])
```

Figure 8: Label Encoding

7.2 Data Sampling

We have to divide the crop dataset and based in this has to be done before we train the model during the model building process. The training set is used to train the model, whereas the testing set is used to evaluate the model's performance on unseen data.The provided code segment is useful for temporally separating the cleaned data frame into training and testing data based on the crop year. In particular, data up to the year 2015 is used for training, with X train including the feature columns (excluding 'Yield') and Y train containing the corresponding 'Yield' values. Conversely, data from the years after 2015 serves as the testing set, with X test capturing the features and Y test catching the 'Yield' vales.

Here we choose Crop Year from 1997 to 2015 for the training data and 2015 to 2020 for test data for training the different models.

```
[ ] X_train, y_train = df_wheat_cleaned[df_wheat_cleaned['Crop_Year'] <= 2015].drop('Yield', axis=1), df_wheat_cleaned[df_wheat_cleaned['Crop_Year'] <= 2015]['Yield']
    X_test, y_test = df_wheat_cleaned[df_wheat_cleaned['Crop_Year'] > 2015].drop('Yield', axis=1), df_wheat_cleaned[df_wheat_cleaned['Crop_Year'] > 2015]['Yield']
```

Figure 9: Data Splitting for Test and Train

7.3 Implementation of Random Forest vs SVM vs Decision Tree

```
[ ] rf_model = RandomForestRegressor(n_estimators=10, random_state=42,max_depth=5)
    svm_model = SVR()
    model_dt = DecisionTreeRegressor(random_state=42)

[ ] rf_model.fit(X_train, y_train)
    svm_model.fit(X_train, y_train)
    model_dt.fit(X_train, y_train)

    DecisionTreeRegressor(random_state=42)

    predictions_rf = rf_model.predict(X_test)
    predictions_svm = svm_model.predict(X_test)
    predictions_dt = model_dt.predict(X_test)
```

Figure 10: Implementation of RF, SVM and Decision Tree

7.4 Implementation Time Series Cross Validation

```
[ ] X = df_wheat_cleaned.drop('Yield', axis=1)
    y = df_wheat_cleaned['Yield']

[ ] tscv = TimeSeriesSplit(n_splits=5)

    acc_rmse_rf = []
    acc_rmse_svm = []
    acc_rmse_dt = []

    # Iterate over each split and train and evaluate the models
    for i, (train_index, test_index) in enumerate(tscv.split(X)):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Train the Random Forest model
        rf_model.fit(X_train, y_train)
        rf_predictions = rf_model.predict(X_test)
        rf_rmse = sqrt(mean_squared_error(y_test, rf_predictions))
        acc_rmse_rf.append(rf_rmse)
        print(f"Split: {i+1}\nTrain range (Year): {X_train['Crop_Year'].min()} - {X_train['Crop_Year'].max()}\nTest range (Year): {X_test['Crop_Year'].min()} - {X_test['Crop_Year'].max()}\nRandom Forest RMSE

        # Train the SVM model
        svm_model.fit(X_train, y_train)
        svm_predictions = svm_model.predict(X_test)
        svm_rmse = sqrt(mean_squared_error(y_test, svm_predictions))
        acc_rmse_svm.append(svm_rmse)
        print(f"Split: {i+1}\nTrain range (Year): {X_train['Crop_Year'].min()} - {X_train['Crop_Year'].max()}\nTest range (Year): {X_test['Crop_Year'].min()} - {X_test['Crop_Year'].max()}\nSVM RMSE: {svm_rms

        # Train the Decision Tree model
        model_dt.fit(X_train, y_train)
        dt_predictions = model_dt.predict(X_test)
        dt_rmse = sqrt(mean_squared_error(y_test, dt_predictions))
        acc_rmse_dt.append(dt_rmse)
        print(f"Split: {i+1}\nTrain range (Year): {X_train['Crop_Year'].min()} - {X_train['Crop_Year'].max()}\nTest range (Year): {X_test['Crop_Year'].min()} - {X_test['Crop_Year'].max()}\nDecision Tree RMSE
```

Figure 11: Time Series Cross Validation

7.5 Hyper Parameter Tuning

```
[ ] def hyperparameter_tuning(model, params, X, y, cv=5):
        gs = GridSearchCV(model, params, cv=cv, scoring='neg_root_mean_squared_error')
        gs.fit(X, y)
        print(f"Best parameters: {gs.best_params_}")
        print(f"Best score: {gs.best_score_}")
        return gs.best_estimator_

[ ] rf_params = {'n_estimators': [50, 100,200],'max_depth': [None,5, 10]}
    best_rf = hyperparameter_tuning(RandomForestRegressor(), rf_params, X_train, y_train)

    Best parameters: {'max_depth': None, 'n_estimators': 200}
    Best score: -0.13769895079757735
```

```
[ ] dt_params = {'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'],
                 'max_depth': [None, 5, 10, 20],
                 'min_samples_split': [2, 5, 10],
                 'min_samples_leaf': [1, 2, 5]}
    best_dt = hyperparameter_tuning(DecisionTreeRegressor(), dt_params, X_train, y_train)

    Best parameters: {'criterion': 'squared_error', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2}
    Best score: -0.1933948377275966

    svm_params = {'kernel':['rbf'],'C': [1, 10, 100,1e3]}
    best_svm = hyperparameter_tuning(SVR(), svm_params, X_train, y_train)

    Best parameters: {'C': 1000.0, 'kernel': 'rbf'}
    Best score: -0.3414279268318744
```

```
    svm_model2 = SVR(kernel='rbf', C=1e3)
    svm_model2.fit(X_train, y_train)
    predictions_svm2 = svm_model2.predict(X_test)
    rmse_svm2 = np.sqrt(mean_squared_error(y_test, predictions_svm2))

[ ] model_dt2 = DecisionTreeRegressor(criterion='poisson', max_depth= 20, min_samples_leaf= 5, min_samples_split= 5,random_state=42)
    model_dt2.fit(X_train, y_train)
    predictions_dt2 = model_dt2.predict(X_test)
    rmse_dt2 = np.sqrt(mean_squared_error(y_test, predictions_dt2))
```

Figure 12: Hyper Parameter tuning

7.6 Ensemble Models - Voting Average

```
▾ Voting Average

    ens_rf = RandomForestRegressor(n_estimators=100, max_depth=None, min_samples_split=2)
    ens_svm = SVR(C=10, kernel='rbf')
    ens_dt = DecisionTreeRegressor(max_depth=None, min_samples_split=2)

    estimators = [
        ('rf', ens_rf),
        ('svm', ens_svm),
        ('dt', ens_dt)
    ]

[ ] voting_reg = VotingRegressor(estimators)

[ ] voting_reg.fit(X_train, y_train)
    voting_preds = voting_reg.predict(X_test)
```

Figure 13: Ensemble - Voting Average

7.7 Ensemble Models - Stacked Generalization

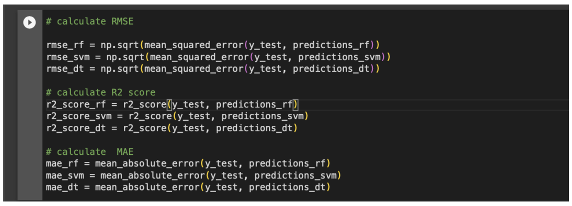
```
▾ Stacked Generalization

[ ] final_estimator = LinearRegression()

[ ] stacking_reg = StackingRegressor(estimators=estimators, final_estimator=final_estimator)
    stacking_reg.fit(X_train, y_train)
    stacking_preds = stacking_reg.predict(X_test)

[ ] stacking_rmse = sqrt(mean_squared_error(y_test, stacking_preds))
```

Figure 14: Ensemble - Stacked Generalization

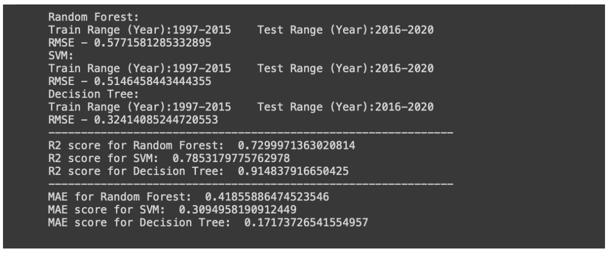
8 Evaluation of Results

The key goal of this study is to predict the wheat yield in India using an ensemble machine learning model and compare it with the performance of the standalone machine learning models like Random Forest(RF),Support Vector Machine(SVM), Decision Tree(DT). The data spanned from 1997 to 2020 and was categorized by state, district, and season. The models were trained using data up to 2015, and their performance was tested on data from 2016 to 2020.

8.1 Results of Models with Default Parameters



```
# calculate RMSE

rmse_rf = np.sqrt(mean_squared_error(y_test, predictions_rf))
rmse_svm = np.sqrt(mean_squared_error(y_test, predictions_svm))
rmse_dt = np.sqrt(mean_squared_error(y_test, predictions_dt))

# calculate R2 score
r2_score_rf = r2_score(y_test, predictions_rf)
r2_score_svm = r2_score(y_test, predictions_svm)
r2_score_dt = r2_score(y_test, predictions_dt)

# calculate  MAE
mae_rf = mean_absolute_error(y_test, predictions_rf)
mae_svm = mean_absolute_error(y_test, predictions_svm)
mae_dt = mean_absolute_error(y_test, predictions_dt)
```



```
Random Forest:
Train Range (Year):1997-2015    Test Range (Year):2016-2020
RMSE - 0.5771581285332895
SVM:
Train Range (Year):1997-2015    Test Range (Year):2016-2020
RMSE - 0.5146458443444355
Decision Tree:
Train Range (Year):1997-2015    Test Range (Year):2016-2020
RMSE - 0.32414085244720553
--------------------------------------------------------------
R2 score for Random Forest:  0.7299971363020814
R2 score for SVM:  0.7853179775762978
R2 score for Decision Tree:  0.914837916650425
--------------------------------------------------------------
MAE for Random Forest:  0.41855886474523546
MAE score for SVM:  0.3094958190912449
MAE score for Decision Tree:  0.17173726541554957
```

Figure 15: Results of Model with
Default parameters

8.2 Results of Time Series Cross Validation

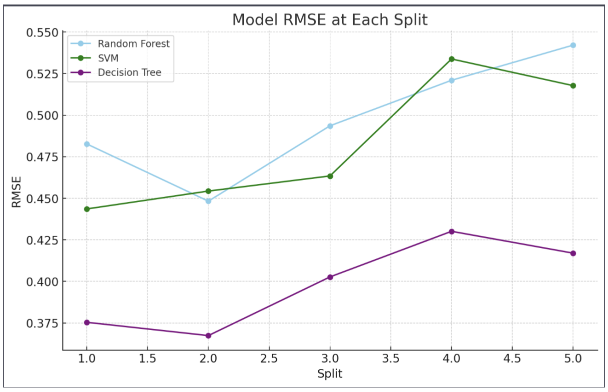
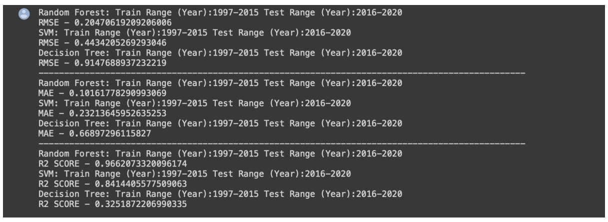


Figure 16: Results of Time series cross validation



```
Random Forest: Train Range (Year):1997-2015 Test Range (Year):2016-2020
RMSE - 0.20470619209206006
SVM: Train Range (Year):1997-2015 Test Range (Year):2016-2020
RMSE - 0.4434205269293046
Decision Tree: Train Range (Year):1997-2015 Test Range (Year):2016-2020
RMSE - 0.9147688937232219
--------------------------------------------------------------------------------
Random Forest: Train Range (Year):1997-2015 Test Range (Year):2016-2020
MAE - 0.10161778290993069
SVM: Train Range (Year):1997-2015 Test Range (Year):2016-2020
MAE - 0.23213645952635253
Decision Tree: Train Range (Year):1997-2015 Test Range (Year):2016-2020
MAE - 0.66897296115827
--------------------------------------------------------------------------------
Random Forest: Train Range (Year):1997-2015 Test Range (Year):2016-2020
R2 SCORE - 0.9662073320096174
SVM: Train Range (Year):1997-2015 Test Range (Year):2016-2020
R2 SCORE - 0.8414405577509063
Decision Tree: Train Range (Year):1997-2015 Test Range (Year):2016-2020
R2 SCORE - 0.3251872206990335
```

Figure 17: RMSE, MAE, R2 for Time Series Cross Validation

8.3 Results of Models After Hyper Parameter Tuning

```
[ ] # calculate RMSE

    rmse_rf = np.sqrt(mean_squared_error(y_test, predictions_rf))
    rmse_svm = np.sqrt(mean_squared_error(y_test, predictions_svm))
    rmse_dt = np.sqrt(mean_squared_error(y_test, predictions_dt))

    # calculate R2 score
    r2_score_rf = r2_score(y_test, predictions_rf)
    r2_score_svm = r2_score(y_test, predictions_svm)
    r2_score_dt = r2_score(y_test, predictions_dt)

    # calculate  MAE
    mae_rf = mean_absolute_error(y_test, predictions_rf)
    mae_svm = mean_absolute_error(y_test, predictions_svm)
    mae_dt = mean_absolute_error(y_test, predictions_dt)
```

```
Random Forest: Train Range (Year):1997-2015 Test Range (Year):2016-2020
RMSE - 0.20470619209206006
SVM: Train Range (Year):1997-2015 Test Range (Year):2016-2020
RMSE - 0.4434205269293046
Decision Tree: Train Range (Year):1997-2015 Test Range (Year):2016-2020
RMSE - 0.9147688937232219
----------------------------------------------------------------------------------------
Random Forest: Train Range (Year):1997-2015 Test Range (Year):2016-2020
MAE - 0.10161778290993069
SVM: Train Range (Year):1997-2015 Test Range (Year):2016-2020
MAE - 0.23213645952635253
Decision Tree: Train Range (Year):1997-2015 Test Range (Year):2016-2020
MAE - 0.66897296115827
----------------------------------------------------------------------------------------
Random Forest: Train Range (Year):1997-2015 Test Range (Year):2016-2020
R2 SCORE - 0.9662073320096174
SVM: Train Range (Year):1997-2015 Test Range (Year):2016-2020
R2 SCORE - 0.8414405577509063
Decision Tree: Train Range (Year):1997-2015 Test Range (Year):2016-2020
R2 SCORE - 0.3251872206990335
```

Figure 18: Results of Models after Tuning

8.4 Results of Ensemble Models

The Voting Average method produced commendable predictions for wheat yield, achieving an RMSE of 0.286, an R^2 score of 0.934, and an MAE of 0.150, indicating its superiority over standalone models used in this research. In contrast, the Stacked Generalization method showcased even more impressive performance with an RMSE of 0.204, an R^2 score of 0.966, and an MAE of 0.101, denoting a notably higher accuracy and capability in accounting for the variance in wheat yield data.

```
[ ] voting_rmse = sqrt(mean_squared_error(y_test, voting_preds))
    # calculate R2 score
    voting_r2_score = r2_score(y_test, voting_preds)
    # #calculate MAE
    voting_mae = mean_absolute_error(y_test, voting_preds)

    print(f"Voting RMSE: {voting_rmse}")
    print(f"Voting R2: {voting_r2_score}")
    print(f"Voting MAE: {voting_mae}")

    Voting RMSE: 0.28565490430937773
    Voting R2: 0.9341972730866314
    Voting MAE: 0.1479194480569795
```

Figure 19: Results of Ensemble - Voting Average

```
[ ] # calculate R2 score
    stacking_r2_score = r2_score(y_test, stacking_preds)
    # #calculate MAE
    stacking_mae = mean_absolute_error(y_test, stacking_preds)

[ ] print(f"Stacking RMSE: {stacking_rmse}")
    print(f"Stacking R2: {stacking_r2_score}")
    print(f"Stacking MAE: {stacking_mae}")

    Stacking RMSE: 0.20716658376865774
    Stacking R2: 0.965390132950514
    Stacking MAE: 0.103413899682968
```

Figure 20: Results of Ensemble - Stacked Generalization