Table of Contents

Chapter No.	Title	Page No.
I	List of Table	03
II	List of Figures	04
III	List of Abbreviations	05
1	Introduction	06
2	Project Description	12
	2.1 Programming Language	14
	2.2 Machine Learning Approaches	14
	2.2.1 Linear Regression	14
	2.2.2 SVR	15
	2.2.3 Artificial Neural Network	15
	2.3 Libraries	16
	2.3.1 Numpy	16
	2.3.2 Pandas	17
	2.3.3 Matplotlib	17
	2.3.4 Seaborn	17
	2.3.5 Keras	18
	2.3.6 Scipy	18
	2.3.7 Scikit-learn	18
3	System Specification	20
	3.1 System Specification	22

	3.1.2 Hardware Description	23
	3.2 Software Specification	24
	3.2.1 Software Description	25
4	Database Description	27
	4.1 Dataset	29
	4.1.1 Dataset Description	29
5	Data Flow Diagram	30
6	Source Code With Results	33
7	Algorithm Comparison Results	70
	7.1 Training on Complete Dataset	72
	7.2 Training on Tamil Nadu Dataset	72
8	Conclusion	74
9	References	76

I. LIST OF TABLES

Table No	Title	Page No
1	Training on Complete Dataset	72
2	Training on Tamil Nadu Dataset	72

II. LIST OF FIGURES

Figure No	Title	Page No
1	DFD	32

III. LIST OF ABBREVIATIONS

NCMRWF	-	National Centre for Medium Range Weather
SST	-	Sea Surface Temperature
MOS	-	Model Output Statistics
ANN	-	Artificial Neural Network
FIS	-	Fuzzy Inference System
ANFIS	-	Adaptive Neuro-Fuzzy Inference System
RMSE	-	Root Mean Square Error
MBE	-	Mean Bias Error
RBFN	-	Radial Basis Function Network
MLP	-	Multi-Layered Perceptron
ERNN	-	Elman Recurrent Neural Network
HFM	-	Hopfield Model
ARIMA	-	Autoregressive Integrated Moving Average
FNN	-	Fuzzy Neural Network
LMP	-	Local Monsoonal Precipitation
SVR	-	Support Vector Regression
SVM	-	Support Vector Machine

Chapter 1

Introduction

Chapter 1

Introduction

Weather forecasting is the application of science and technology to predict the conditions of the atmosphere for a given location and time. People have attempted to predict the weather informally for millennia and formally since the 19th century. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere at a given place and using meteorology to project how the atmosphere will change.

Ancient weather forecasting methods usually relied on observed patterns of events, also termed pattern recognition. For example, it might be observed that if the sunset was particularly red, the following day often brought fair weather. This experience accumulated over the generations to produce weather lore.

Once calculated by hand based mainly upon changes in barometric pressure, current weather conditions, and sky condition or cloud cover, weather forecasting now relies on computer-based models that take many atmospheric factors into account. Human input is still required to pick the best possible forecast model to base the forecast upon, which involves pattern recognition skills, teleconnections, knowledge of model performance, and knowledge of model biases. The inaccuracy of forecasting is due to the chaotic nature of the atmosphere, the massive computational power required to solve the equations that describe the atmosphere, the error involved in measuring the initial conditions, and an incomplete understanding of atmospheric processes. Hence, forecasts become less accurate as the difference between current time and the time for which the forecast is being made (the range of the forecast) increases. The use of ensembles and model consensus help narrow the error and pick the most likely outcome.

Humans are required to interpret the model data into weather forecasts that are understandable to the end user. Humans can use knowledge of local effects that may be too small in size to be resolved by the model to add information to the forecast. While increasing accuracy of forecast models implies that humans may no longer be needed in the forecast process at some point in the future, there is currently still a need for human intervention.

Digital computers have made it possible to calculate changes in atmospheric conditions mathematically and objectively—i.e., in such a way that anyone can obtain the same result from the same initial conditions. The widespread adoption of numerical weather prediction models brought a whole new group of players—computer specialists and experts in numerical processing and statistics—to the scene to work with atmospheric scientists and meteorologists. Moreover, the enhanced capability to process and analyse weather data stimulated the long-standing interest of meteorologists in securing more observations of greater accuracy. Technological advances since the 1960s led to a growing reliance on remote sensing, particularly the gathering of data with specially instrumented Earth-orbiting satellites. By the late 1980s, forecasts of the weather were largely based on the determinations of numerical models integrated by high-speed supercomputers—except for some shorter-range predictions, particularly those related to local thunderstorm activity, which were made by specialists directly interpreting radar and satellite measurements. By the early 1990s a network of next-generation Doppler weather radar was largely in place in the United States, which allowed meteorologists to predict severe weather events with additional lead time before their occurrence.

There are a variety of end uses to weather forecasts. Weather warnings are important forecasts because they are used to protect life and property. Forecasts based on temperature and precipitation are important to agriculture, and therefore to traders within commodity markets. Temperature forecasts are used by utility companies to estimate demand over coming days. On an everyday basis, people use weather forecasts to determine what to wear on a given day. Since outdoor activities are severely curtailed by heavy rain, snow and wind chill, forecasts can be used to plan activities around these events, and to plan ahead and survive them. In 2009, the US spent $5.1 billion on weather forecasting.

In this procedure a day-to-day analysis of the weather is to be predicted and these forecasting should be communicated for the end users for taking decisions. It is a most challenging issue since the decisions that are taken are mostly with uncertainty. Researchers in this domain have categorized these forecasting strategies into a 2-fold, based on numeric modelling and based scientific processing. Among these categorizations, the predictions of the rainfalls are mostly subjected to the numerical methods of analysis. Since the data changes dynamically, in particular with rainfall/weather, results into uncertainties in most of the cases. This is due to the choice of the initial conditions which are mainly used for weather predictions and which are viable of changing. Therefore, to update these methods statistical techniques have been used at a high pace.

Among these models time-series models are mostly considered to understand and interpret the patterns of the rainfalls. However, these statistical methods suffer from disadvantages like existing of non-linear relationships between the monsoon data and the prediction data and secondly for evaluating the analysis, these statistical models needs to identify the predictions more effectively. Therefore, identifying these parameters has become a challenging task. Many researchers have highlighted that most of the changes in the climatic conditions are mostly due to the global weather changes. Therefore, if these weather changes are identified, effective prediction techniques can be planned.

Data mining techniques play a vital role in the prediction of global parameters with these assumptions, the knowledge discovery is carried out by integrating the time series analysing together with data mining techniques.

Artificial neural networks also help in effective forecasting as it can handle the non-linearity cases arose from the statistical models.

Chapter 2

Project Description

Chapter 2

Project Description

2.1 Programming Language

Python is a dynamic scripting language similar to Perl and Ruby. The principal author of Python is Guido van Rossum. Python supports dynamic typing and has a garbage collector for automatic memory management. Another important feature of Python is dynamic name solution which binds the names of functions and variables during execution

2.2 Machine Learning Approaches

We apply various statistical and machine learning approaches such as

- Linear Regression
- SVR
- Artificial Neural Network

in prediction and make analysis over various approaches. By using various approaches, we try to minimize the error.

Testing metrics: We used Mean absolute error to train the models.

2.2.1 Linear Regression

Linear regression is probably one of the most important and widely used regression techniques. It’s among the simplest regression methods. One of its main advantages is the ease of interpreting results.

It is a very powerful technique and can be used to understand the factors that influence profitability. It can be used to forecast sales in the coming months by analyzing the sales data for previous months. It can also be used to gain various insights about customer behaviour.

2.2.2 SVR

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

2.2.3 Artificial Neural Network

Artificial Neural Networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labelled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process.

2.3 Libraries

We use the following libraries in the project

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Keras
- Scipy
- Scikit-learn

2.3.1NumPy

NumPy is the fundamental package for array computing with Python.

It provides:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- and much more

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2.3.2 Pandas

Powerful data structures for data analysis, time series, and statistics

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

2.3.3 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

2.3.4 Seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of Matplotlib and closely integrated with Pandas data structures.

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

2.3.5 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

2.3.6 Scipy

SciPy (pronounced “Sigh Pie”) is open-source software for mathematics, science, and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install, and are free of charge. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world’s leading scientists and engineers.

2.3.7 Scikit-learn

Scikit-learn is a set of python modules for machine learning and data mining

It is an open source Python library that implements a range of machine learning, pre processing, cross-validation and visualization algorithms using a unified interface.

Chapter 3

System Specification

3. SYSTEM SPECIFICATION

3.1. HARDWARE SPECIFICATION

Processor	: Intel Core i3 processor
Memory	: 4.00 GB
Hard Disk	: 500 GB
Monitor	: Monitor with 1920x1080 Resolution
Processor Speed	: 2.50 GHz

3.1.2 Hardware Description

The Intel Core i3 processor was developed and manufactured by Intel. It is a dual-core computer processor. It is one of three types of processors in the "i" series (also called the Intel Core family of processors). The Core i3 processor is available in multiple speeds, ranging from 1.30 GHz up to 3.50 GHz, and features either 3 MB or 4 MB of cache. It utilizes either the LGA 1150 or LGA 1155 socket on a motherboard. Core i3 processors are most often found as dual-core, having two cores. However, a select few high-end Core i3 processors are quad-core, featuring four cores.

4Gb RAM, offers increased speed and efficiency for computing devices. This new memory chip technology boasts increased transfer speeds that will boost performance of your favourite devices.

3.2 Software Specification

Operating System	: Windows 8.1 Pro
System type	: 64-bit Operating System, x64 based Processor
IDE	: Anaconda
	: Spyder (Anaconda)
	: Jupyter Notebook (Anaconda)

3.2.1 Software Description

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012.

Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrate with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open source software. It is released under the MIT license. Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

Chapter 4

Database Description

Chapter 4

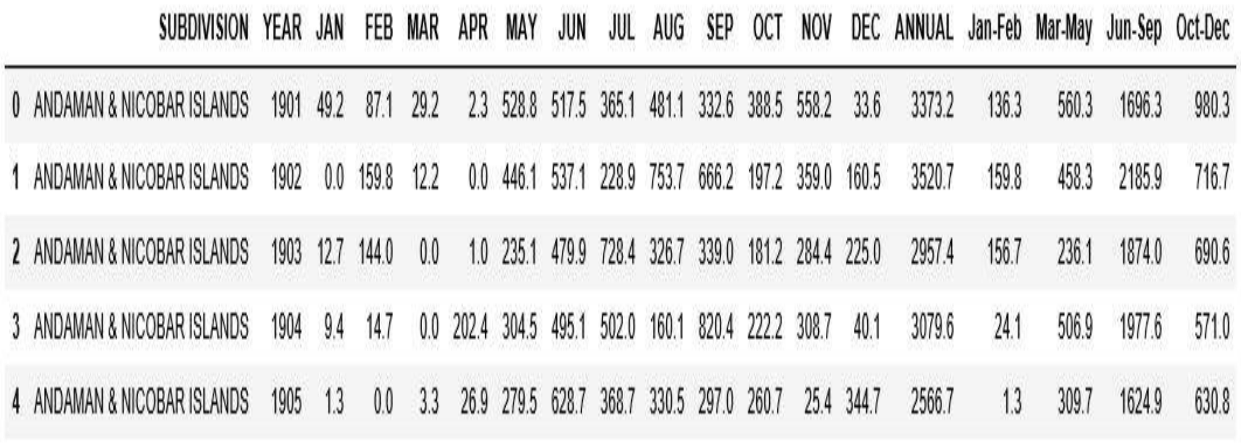
Database Description

4.1 Dataset

This dataset has average rainfall from 1951-2000 for each district, for every month.

4.1.1 Dataset Description

- Data has 36 sub divisions and 19 attributes (individual months, annual, combinations of 3 consecutive months).
- For some of the subdivisions data is from 1950 to 2015.
- All the attributes have the sum of amount of rainfall in mm.



	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
0	ANDAMAN & NICOBAR ISLANDS	1901	49.2	87.1	29.2	2.3	528.8	517.5	365.1	481.1	332.6	388.5	558.2	33.6	3373.2	136.3	560.3	1696.3	980.3
1	ANDAMAN & NICOBAR ISLANDS	1902	0.0	159.8	12.2	0.0	446.1	537.1	228.9	753.7	666.2	197.2	359.0	160.5	3520.7	159.8	458.3	2185.9	716.7
2	ANDAMAN & NICOBAR ISLANDS	1903	12.7	144.0	0.0	1.0	235.1	479.9	728.4	326.7	339.0	181.2	284.4	225.0	2957.4	156.7	236.1	1874.0	690.6
3	ANDAMAN & NICOBAR ISLANDS	1904	9.4	14.7	0.0	202.4	304.5	495.1	502.0	160.1	820.4	222.2	308.7	40.1	3079.6	24.1	506.9	1977.6	571.0
4	ANDAMAN & NICOBAR ISLANDS	1905	1.3	0.0	3.3	26.9	279.5	628.7	368.7	330.5	297.0	260.7	25.4	344.7	2566.7	1.3	309.7	1624.9	630.8

Chapter 5

Data Flow Diagram

CHAPTER 5

DATA FLOW DIAGRAM

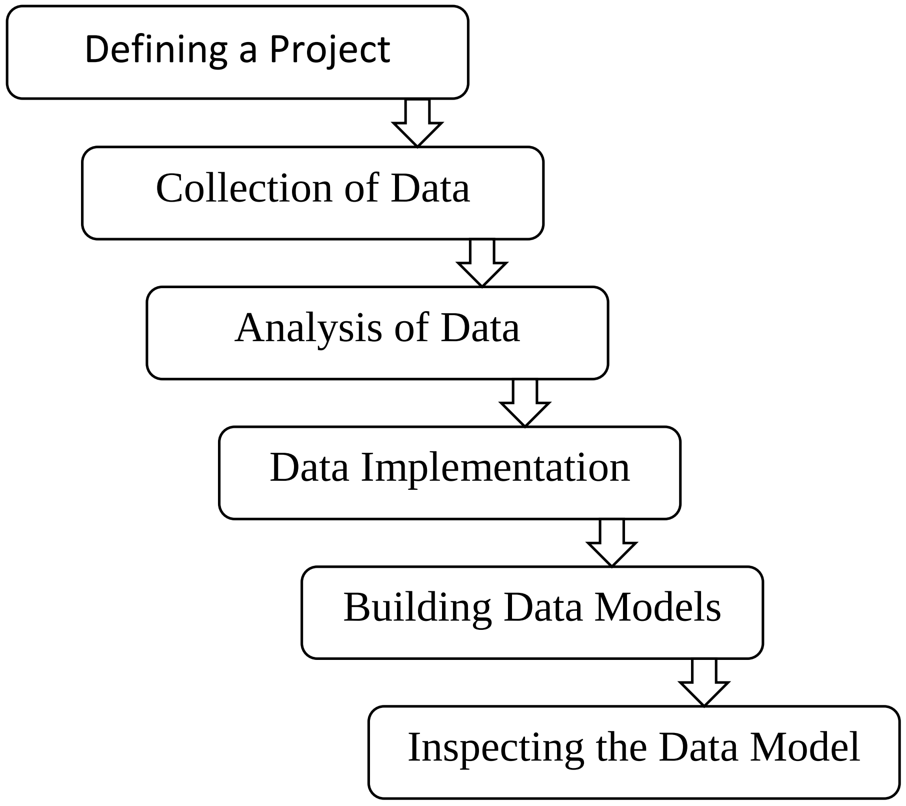


Fig 1: DFD

Chapter 6

Source Code with Results

CHAPTER 6

SOURCE CODE WITH RESULTS

In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
```

Types of graphs

- Bar graphs showing distribution of amount of rainfall.
- Distribution of amount of rainfall yearly, monthly, groups of months.
- Distribution of rainfall in subdivisions, districts form each month, groups of months.
- Heat maps showing correlation between amount of rainfall between months.

In [2]:

```
data = pd.read_csv("C:/Users/Vinoth Kumar R/Downloads/Telegram Desktop/rainfall-predict
ion-master/rainfall-prediction-master/data/rainfall_in_india_1901-2015.csv",sep=",")
data = data.fillna(data.mean())
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4116 entries, 0 to 4115
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   SUBDIVISION  4116 non-null   object 
 1   YEAR         4116 non-null   int64  
 2   JAN          4116 non-null   float64
 3   FEB          4116 non-null   float64
 4   MAR          4116 non-null   float64
 5   APR          4116 non-null   float64
 6   MAY          4116 non-null   float64
 7   JUN          4116 non-null   float64
 8   JUL          4116 non-null   float64
 9   AUG          4116 non-null   float64
 10  SEP          4116 non-null   float64
 11  OCT          4116 non-null   float64
 12  NOV          4116 non-null   float64
 13  DEC          4116 non-null   float64
 14  ANNUAL       4116 non-null   float64
 15  Jan-Feb      4116 non-null   float64
 16  Mar-May      4116 non-null   float64
 17  Jun-Sep      4116 non-null   float64
 18  Oct-Dec      4116 non-null   float64
dtypes: float64(17), int64(1), object(1)
memory usage: 611.1+ KB
```

Dataset Description

- Data has 36 sub divisions and 19 attributes (individual months, annual, combinations of 3 consecutive months).
- For some of the subdivisions data is from 1950 to 2015.
- All the attributes has the sum of amount of rainfall in mm.

In [3]:

```
data.head()
```

Out[3]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT
0	ANDAMAN & NICOBAR ISLANDS	1901	49.2	87.1	29.2	2.3	528.8	517.5	365.1	481.1	332.6	388.5
1	ANDAMAN & NICOBAR ISLANDS	1902	0.0	159.8	12.2	0.0	446.1	537.1	228.9	753.7	666.2	197.2
2	ANDAMAN & NICOBAR ISLANDS	1903	12.7	144.0	0.0	1.0	235.1	479.9	728.4	326.7	339.0	181.2
3	ANDAMAN & NICOBAR ISLANDS	1904	9.4	14.7	0.0	202.4	304.5	495.1	502.0	160.1	820.4	222.2
4	ANDAMAN & NICOBAR ISLANDS	1905	1.3	0.0	3.3	26.9	279.5	628.7	368.7	330.5	297.0	260.7

In [4]:

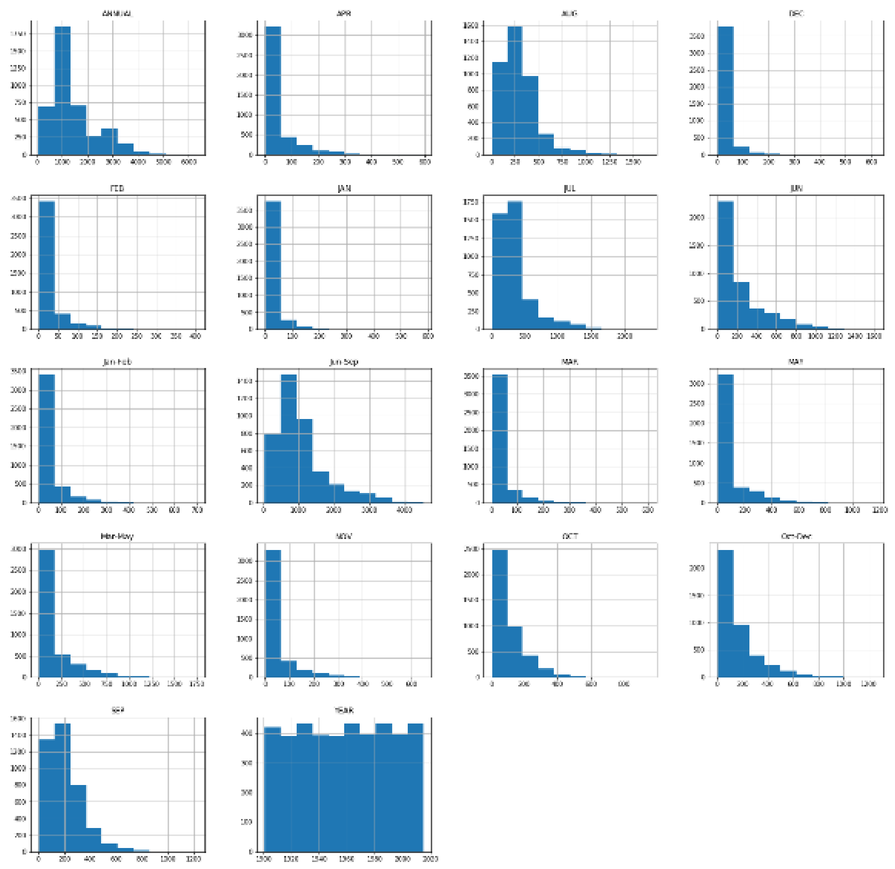
```
data.describe()
```

Out[4]:

	YEAR	JAN	FEB	MAR	APR	MAY	
count	4116.000000	4116.000000	4116.000000	4116.000000	4116.000000	4116.000000	4116.0
mean	1958.218659	18.957320	21.805325	27.359197	43.127432	85.745417	230.2
std	33.140898	33.569044	35.896396	46.925176	67.798192	123.189974	234.5
min	1901.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.4
25%	1930.000000	0.600000	0.600000	1.000000	3.000000	8.600000	70.4
50%	1958.000000	6.000000	6.700000	7.900000	15.700000	36.700000	138.9
75%	1987.000000	22.125000	26.800000	31.225000	49.825000	96.825000	304.9
max	2015.000000	583.700000	403.500000	605.600000	595.100000	1168.600000	1609.9

In [5]:

```
data.hist(figsize=(24,24));
```

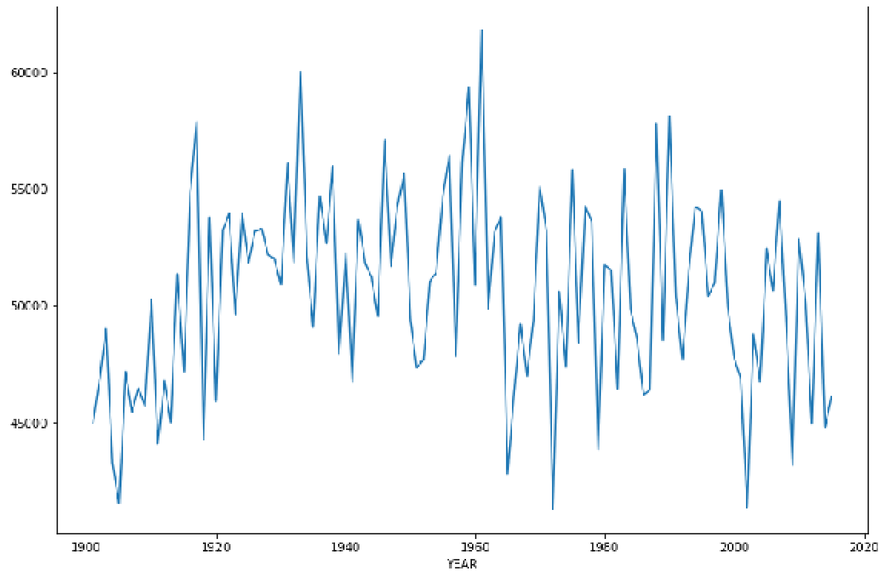


Observations

- Above histograms show the distribution of rainfall over months.
- Observed increase in amount of rainfall over months July, August, September.

In [6]:

```
data.groupby("YEAR").sum()['ANNUAL'].plot(figsize=(12,8));
```

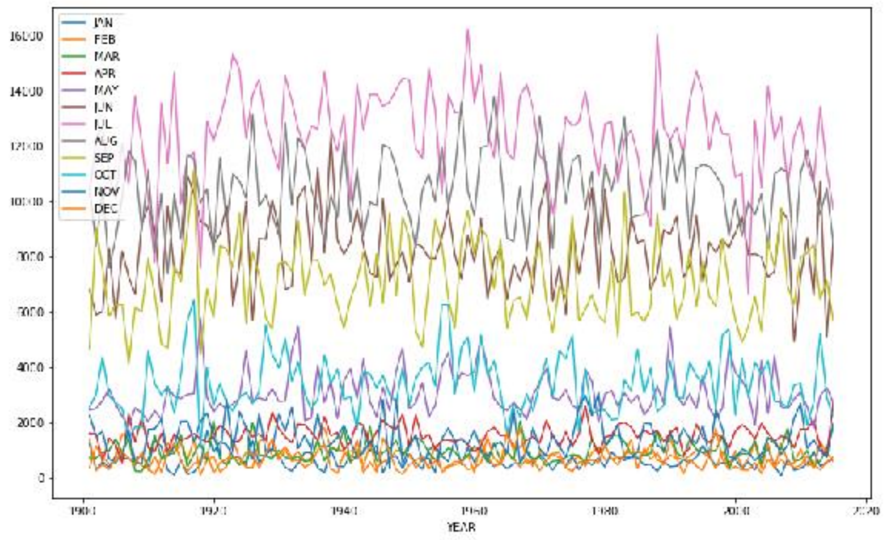


Observations

- Shows distribution of rainfall over years.
- Observed high amount of rainfall in 1950s.

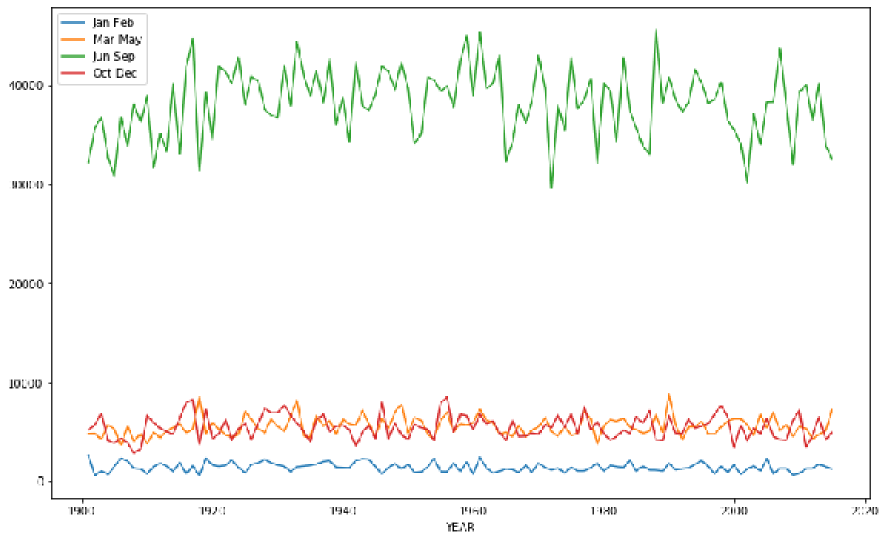
```
In [7]:
```

```
data[['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
      'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("YEAR").sum().plot(figsize=(13,8));
```



In [8]:

```
data[['YEAR','Jan-Feb', 'Mar-May',
       'Jun-Sep', 'Oct-Dec']].groupby("YEAR").sum().plot(figsize=(13,8));
```

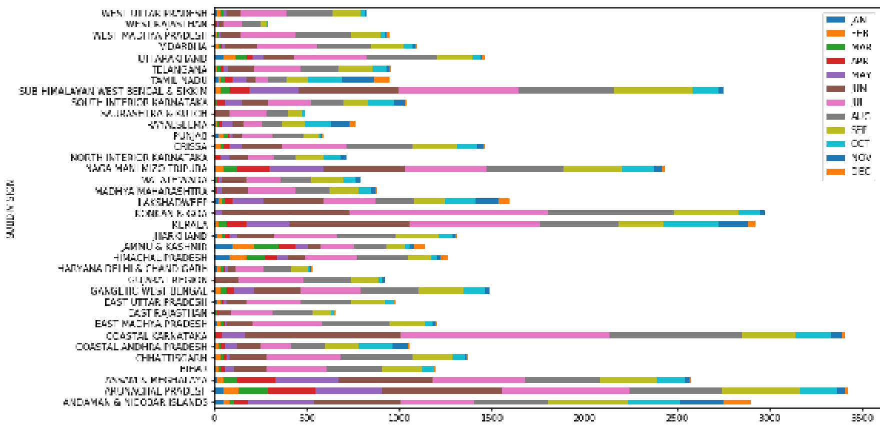


Observations

- The above two graphs show the distribution of rainfall over months.
- The graphs clearly shows that amount of rainfall in high in the months july, aug, sep which is monsoon season in India.

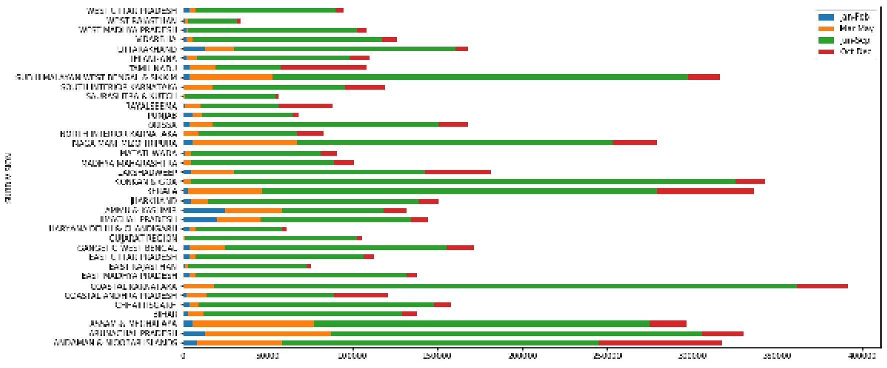
In [9]:

```
data[['SUBDIVISION', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("SUBDIVISION").mean().plot.barh(stac
ked=True,figsize=(13,8));
```



In [10]:

```
data[['SUBDIVISION', 'Jan-Feb', 'Mar-May',
       'Jun-Sep', 'Oct-Dec']].groupby("SUBDIVISION").sum().plot.barh(stacked=True,figsi
ze=(16,8));
```

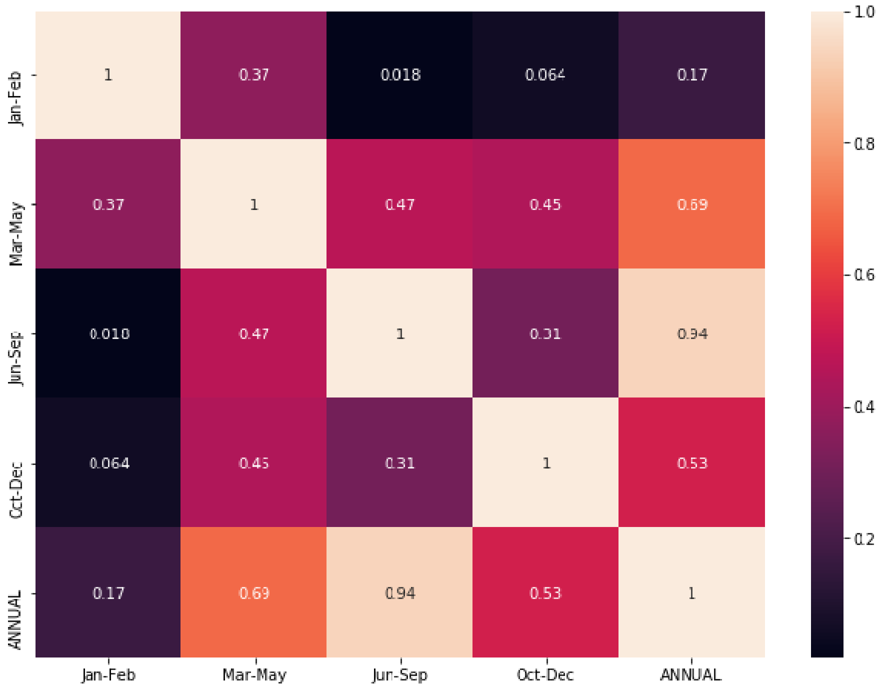


Observations

- Above two graphs shows that the amount of rainfall is reasonably good in the months of march, april, may in eastern India.

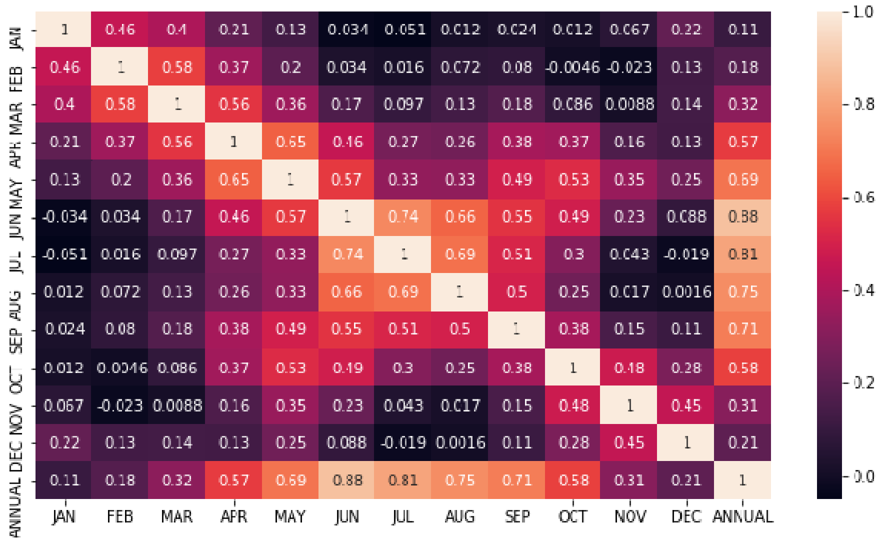
In [11]:

```
plt.figure(figsize=(11,8))
sns.heatmap(data[['Jan-Feb','Mar-May','Jun-Sep','Oct-Dec','ANNUAL']].corr(),annot=True)
plt.show()
```



```
In [12]:
```

```
plt.figure(figsize=(11,6))
sns.heatmap(data[['JAN','FEB','MAR','APR','MAY','JUN','JUL','AUG','SEP','OCT','NOV','DE
C','ANNUAL']].corr(),annot=True)
plt.show()
```



	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL
JAN	1	0.46	0.4	0.21	0.13	0.034	0.051	0.012	0.024	0.012	0.067	0.22	0.11
FEB	0.46	1	0.58	0.37	0.2	0.034	0.016	0.072	0.08	-0.0046	-0.023	0.13	0.18
MAR	0.4	0.58	1	0.56	0.36	0.17	0.097	0.13	0.18	0.086	0.0088	0.14	0.32
APR	0.21	0.37	0.56	1	0.65	0.46	0.27	0.26	0.38	0.37	0.16	0.13	0.57
MAY	0.13	0.2	0.36	0.65	1	0.57	0.33	0.33	0.49	0.53	0.35	0.25	0.69
JUN	-0.034	0.034	0.17	0.46	0.57	1	0.74	0.66	0.55	0.49	0.23	0.088	0.88
JUL	-0.051	0.016	0.097	0.27	0.33	0.74	1	0.69	0.51	0.3	0.043	-0.019	0.81
AUG	0.012	0.072	0.13	0.26	0.33	0.66	0.69	1	0.5	0.25	0.017	0.0016	0.75
SEP	0.024	0.08	0.18	0.38	0.49	0.55	0.51	0.5	1	0.38	0.15	0.11	0.71
OCT	0.012	0.0046	0.086	0.37	0.53	0.49	0.3	0.25	0.38	1	0.48	0.28	0.58
NOV	0.067	-0.023	0.0088	0.16	0.35	0.23	0.043	0.017	0.15	0.48	1	0.45	0.31
DEC	0.22	0.13	0.14	0.13	0.25	0.088	-0.019	0.0016	0.11	0.28	0.45	1	0.21
ANNUAL	0.11	0.18	0.32	0.57	0.69	0.88	0.81	0.75	0.71	0.58	0.31	0.21	1

Observations

- **Heat Map** shows the co-relation(dependency) betwenn the amounts of rainfall over months.
- From above it is clear that if amount of rainfall is high in the months of july, august, september then the amount of rainfall will be high annually.
- It is also obwserved that if amount of rainfall in good in the months of october, november, december then the rainfall is going to b good in the overall year.

In [13]:

```
#Function to plot the graphs
def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)  # the x locations for the groups
    width = 0.35       # the width of the bars

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall")
    ax.set_xticks(ind+width)
    ax.set_xticklabels( ('APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC')
)
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

#     autolabel(rects1)
    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
#     autolabel(rects2)

    plt.show()
```

Predictions

- For prediction we formatted data in the way, given the rainfall in the last three months we try to predict the rainfall in the next consecutive month.
- For all the experiments we used 80:20 training and test ratio.
  - Linear regression
  - SVR
  - Artificial neural nets
- Tersting metrics: We used Mean absolute error to train the models.
- We also shown the amount of rainfall actually and predicted with the histogram plots.
- We did two types of trainings once training on complete dataset and other with training with only tamilnadu data
- All means are standard deviation observations are written, first one represents ground truth, second one represents predictions.

In [14]:

```
# seperation of training and testing data
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=4
2)
```

In [15]:

```
#test 2010
temp = data[['SUBDIVISION','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['YEAR'] == 2010]

data_2010 = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['SUBDIVISION'] == 'TAMIL NADU'])

X_year_2010 = None; y_year_2010 = None
for i in range(data_2010.shape[1]-3):
    if X_year_2010 is None:
        X_year_2010 = data_2010[:, i:i+3]
        y_year_2010 = data_2010[:, i+3]
    else:
        X_year_2010 = np.concatenate((X_year_2010, data_2010[:, i:i+3]), axis=0)
        y_year_2010 = np.concatenate((y_year_2010, data_2010[:, i+3]), axis=0)
```

In [16]:

```
#test 2005
temp = data[['SUBDIVISION','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['YEAR'] == 2005]

data_2005 = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['SUBDIVISION'] == 'TAMIL NADU'])

X_year_2005 = None; y_year_2005 = None
for i in range(data_2005.shape[1]-3):
    if X_year_2005 is None:
        X_year_2005 = data_2005[:, i:i+3]
        y_year_2005 = data_2005[:, i+3]
    else:
        X_year_2005 = np.concatenate((X_year_2005, data_2005[:, i:i+3]), axis=0)
        y_year_2005 = np.concatenate((y_year_2005, data_2005[:, i+3]), axis=0)
```

In [17]:

```
#test 2015
temp = data[['SUBDIVISION','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['YEAR'] == 2015]

data_2015 = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['SUBDIVISION'] == 'TAMIL NADU'])

X_year_2015 = None; y_year_2015 = None
for i in range(data_2015.shape[1]-3):
    if X_year_2015 is None:
        X_year_2015 = data_2015[:, i:i+3]
        y_year_2015 = data_2015[:, i+3]
    else:
        X_year_2015 = np.concatenate((X_year_2015, data_2015[:, i:i+3]), axis=0)
        y_year_2015 = np.concatenate((y_year_2015, data_2015[:, i+3]), axis=0)
```

In [18]:

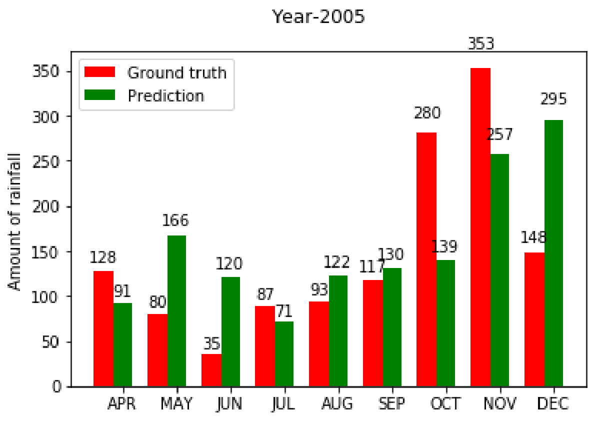
```
from sklearn import linear_model

# linear model
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print(mean_absolute_error(y_test, y_pred))
```

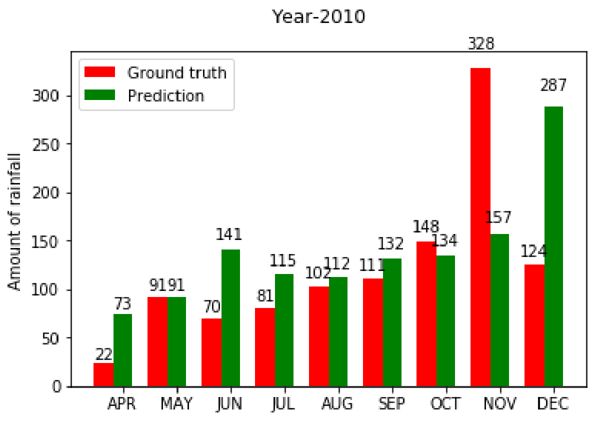
```
96.32435229744084
```

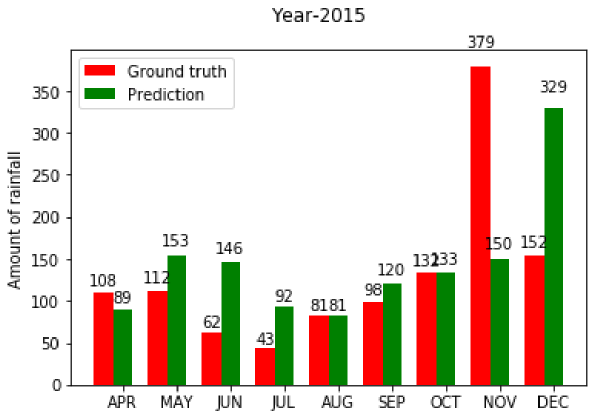
In [19]:

```
#2005
y_year_pred_2005 = reg.predict(X_year_2005)


#2010
y_year_pred_2010 = reg.predict(X_year_2010)

y_year_pred_2015 = reg.predict(X_year_2015)

print("MEAN 2005")
print(np.mean(y_year_2005),np.mean(y_year_pred_2005))
print("Standard deviation 2005")
print(np.sqrt(np.var(y_year_2005)),np.sqrt(np.var(y_year_pred_2005)))


print("MEAN 2010")
print(np.mean(y_year_2010),np.mean(y_year_pred_2010))
print("Standard deviation 2010")
print(np.sqrt(np.var(y_year_2010)),np.sqrt(np.var(y_year_pred_2010)))


print("MEAN 2015")
print(np.mean(y_year_2015),np.mean(y_year_pred_2015))
print("Standard deviation 2015")
print(np.sqrt(np.var(y_year_2015)),np.sqrt(np.var(y_year_pred_2015)))


plot_graphs(y_year_2005,y_year_pred_2005,"Year-2005")
plot_graphs(y_year_2010,y_year_pred_2010,"Year-2010")
plot_graphs(y_year_2015,y_year_pred_2015,"Year-2015")
```

```
MEAN 2005
147.3111111111111 154.89409610331137
Standard deviation 2005
97.07130763223547 70.26461054053196
MEAN 2010
120.15555555555557 138.52491051795914
Standard deviation 2010
80.8999397213223 58.06819529310172
MEAN 2015
130.25555555555556 144.14608860074196
Standard deviation 2015
93.7362625325986 70.61860433793966
```







In [20]:

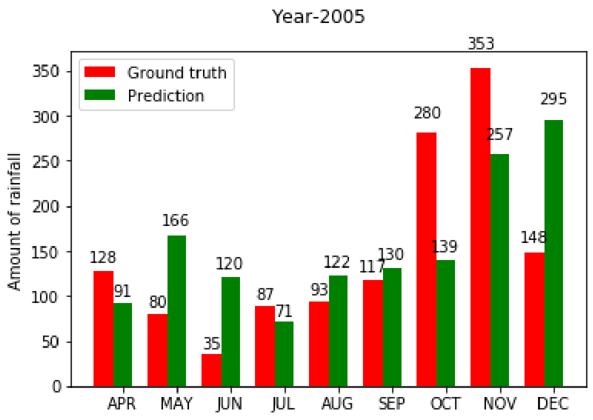
```
from sklearn.svm import SVR

# SVM model
clf = SVR(gamma='auto', C=0.1, epsilon=0.2)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(mean_absolute_error(y_test, y_pred))
```

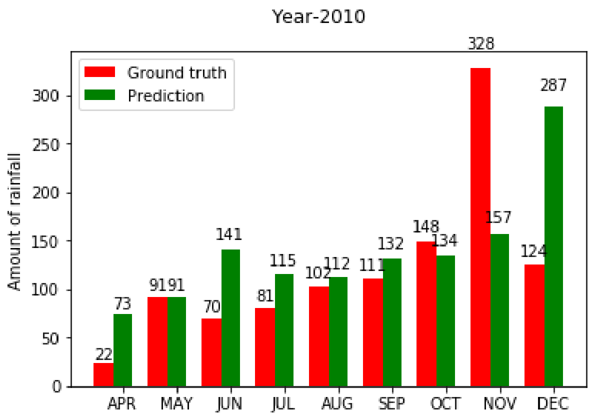
```
127.1600615632603
```

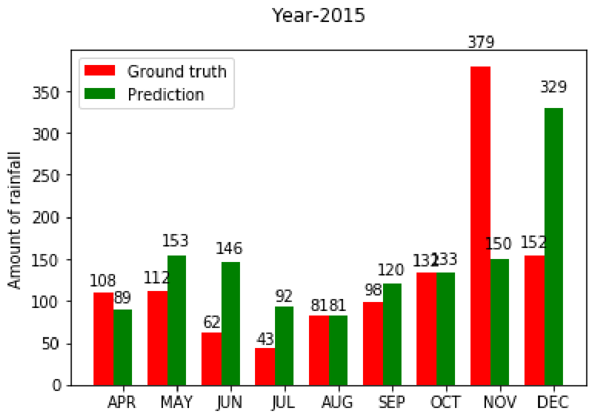
In [21]:

```
#2005
y_year_pred_2005 = reg.predict(X_year_2005)

#2010
y_year_pred_2010 = reg.predict(X_year_2010)

#2015
y_year_pred_2015 = reg.predict(X_year_2015)
print("MEAN 2005")
print(np.mean(y_year_2005),np.mean(y_year_pred_2005))
print("Standard deviation 2005")
print(np.sqrt(np.var(y_year_2005)),np.sqrt(np.var(y_year_pred_2005)))


print("MEAN 2010")
print(np.mean(y_year_2010),np.mean(y_year_pred_2010))
print("Standard deviation 2010")
print(np.sqrt(np.var(y_year_2010)),np.sqrt(np.var(y_year_pred_2010)))


print("MEAN 2015")
print(np.mean(y_year_2015),np.mean(y_year_pred_2015))
print("Standard deviation 2015")
print(np.sqrt(np.var(y_year_2015)),np.sqrt(np.var(y_year_pred_2015)))

plot_graphs(y_year_2005,y_year_pred_2005,"Year-2005")
plot_graphs(y_year_2010,y_year_pred_2010,"Year-2010")
plot_graphs(y_year_2015,y_year_pred_2015,"Year-2015")
```

```
MEAN 2005
147.3111111111111 154.89409610331137
Standard deviation 2005
97.07130763223547 70.26461054053196
MEAN 2010
120.15555555555557 138.52491051795914
Standard deviation 2010
80.8999397213223 58.06819529310172
MEAN 2015
130.25555555555556 144.14608860074196
Standard deviation 2015
93.7362625325986 70.61860433793966
```






Year-2015
Ground truth
Prediction
Amount of rainfall
0
50
100
150
200
250
300
350
108
89
112
153
62
146
43
92
81
81
98
120
132
133
379
150
152
329
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC

In [22]:

```
from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten

# NN model
inputs = Input(shape=(3,1))
x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
x = Conv1D(128, 2, padding='same', activation='elu')(x)
x = Flatten()(x)
x = Dense(128, activation='elu')(x)
x = Dense(64, activation='elu')(x)
x = Dense(32, activation='elu')(x)
x = Dense(1, activation='linear')(x)
model = Model(inputs=[inputs], outputs=[x])
model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
model.summary()
```

```
Using TensorFlow backend.
F:\Anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:516: F
utureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
F:\Anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:517: F
utureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
F:\Anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:518: F
utureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
F:\Anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:519: F
utureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
F:\Anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:520: F
utureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
F:\Anaconda\lib\site-packages\tensorflow\python\framework\dtypes.py:525: F
utureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
F:\Anaconda\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.p
y:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
F:\Anaconda\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.p
y:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
F:\Anaconda\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.p
y:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
F:\Anaconda\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.p
y:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
F:\Anaconda\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.p
y:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
F:\Anaconda\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.p
y:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 3, 1)              0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 3, 64)             192
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 3, 128)            16512
_________________________________________________________________
flatten_1 (Flatten)          (None, 384)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               49280
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 76,353
Trainable params: 76,353
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

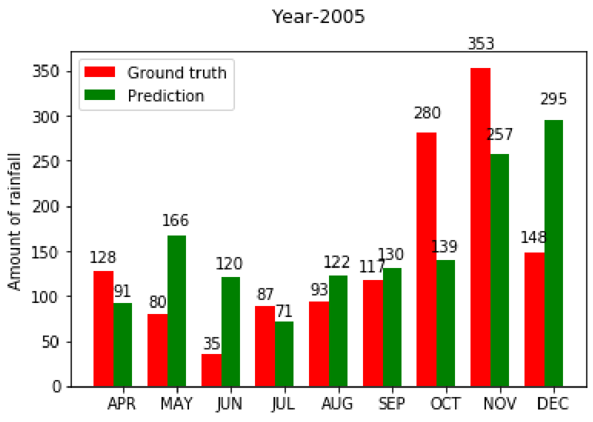
```
model.fit(x=np.expand_dims(X_train, axis=2), y=y_train, batch_size=64, epochs=10, verbo
se=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(X_test, axis=2))
print(mean_absolute_error(y_test, y_pred))
```

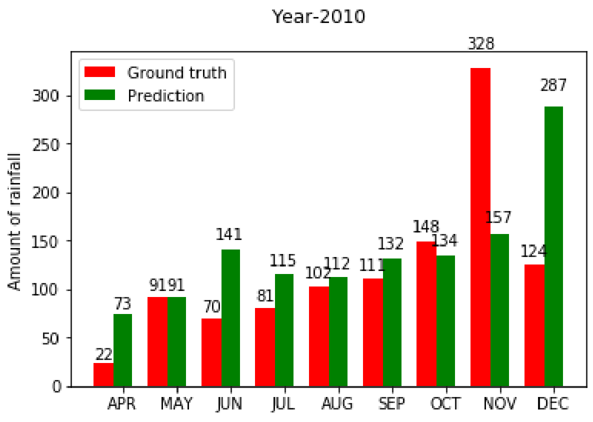
```
WARNING:tensorflow:From F:\Anaconda\lib\site-packages\keras\backend\tensor
flow_backend.py:422: The name tf.global_variables is deprecated. Please us
e tf.compat.v1.global_variables instead.

Train on 30005 samples, validate on 3334 samples
Epoch 1/10
30005/30005 [==============================] - 7s 235us/step - loss: 1915
8.6402 - mae: 87.1812 - val_loss: 17436.5435 - val_mae: 84.3842
Epoch 2/10
30005/30005 [==============================] - 5s 165us/step - loss: 1858
2.7235 - mae: 86.4857 - val_loss: 18008.5904 - val_mae: 90.0884
Epoch 3/10
30005/30005 [==============================] - 5s 158us/step - loss: 1835
7.9601 - mae: 86.1600 - val_loss: 17211.7193 - val_mae: 84.8119
Epoch 4/10
30005/30005 [==============================] - 5s 163us/step - loss: 1830
2.9167 - mae: 85.9506 - val_loss: 17293.0629 - val_mae: 85.8243
Epoch 5/10
30005/30005 [==============================] - 5s 159us/step - loss: 1831
5.7979 - mae: 85.9323 - val_loss: 17316.0606 - val_mae: 83.8115
Epoch 6/10
30005/30005 [==============================] - 5s 163us/step - loss: 1819
0.4314 - mae: 85.4565 - val_loss: 17380.2122 - val_mae: 87.1823
Epoch 7/10
30005/30005 [==============================] - 13s 420us/step - loss: 1812
8.4210 - mae: 85.3049 - val_loss: 17038.8091 - val_mae: 82.7383
Epoch 8/10
30005/30005 [==============================] - 5s 157us/step - loss: 1809
2.2152 - mae: 85.0946 - val_loss: 17274.3974 - val_mae: 87.1517
Epoch 9/10
30005/30005 [==============================] - 5s 164us/step - loss: 1796
9.9528 - mae: 85.1536 - val_loss: 17084.4924 - val_mae: 81.1123
Epoch 10/10
30005/30005 [==============================] - 5s 166us/step - loss: 1790
3.5980 - mae: 84.5953 - val_loss: 17054.8033 - val_mae: 83.6203
86.64192137579357
```

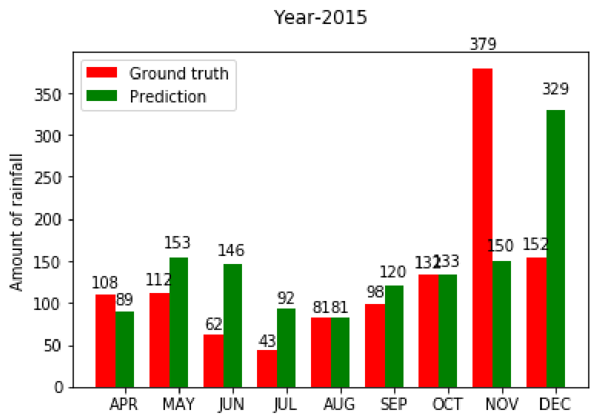
In [24]:

```
#2005
y_year_pred_2005 = reg.predict(X_year_2005)

#2010
y_year_pred_2010 = reg.predict(X_year_2010)

#2015
y_year_pred_2015 = reg.predict(X_year_2015)

print("MEAN 2005")
print(np.mean(y_year_2005),np.mean(y_year_pred_2005))
print("Standard deviation 2005")
print(np.sqrt(np.var(y_year_2005)),np.sqrt(np.var(y_year_pred_2005)))


print("MEAN 2010")
print(np.mean(y_year_2010),np.mean(y_year_pred_2010))
print("Standard deviation 2010")
print(np.sqrt(np.var(y_year_2010)),np.sqrt(np.var(y_year_pred_2010)))


print("MEAN 2015")
print(np.mean(y_year_2015),np.mean(y_year_pred_2015))
print("Standard deviation 2015")
print(np.sqrt(np.var(y_year_2015)),np.sqrt(np.var(y_year_pred_2015)))

plot_graphs(y_year_2005,y_year_pred_2005,"Year-2005")
plot_graphs(y_year_2010,y_year_pred_2010,"Year-2010")
plot_graphs(y_year_2015,y_year_pred_2015,"Year-2015")
```

```
MEAN 2005
147.3111111111111 154.89409610331137
Standard deviation 2005
97.07130763223547 70.26461054053196
MEAN 2010
120.15555555555557 138.52491051795914
Standard deviation 2010
80.8999397213223 58.06819529310172
MEAN 2015
130.25555555555556 144.14608860074196
Standard deviation 2015
93.7362625325986 70.61860433793966
```







In [25]:

```
# spliting training and testing data only for tamilnadu
tamilnadu = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['SUBDIVISION'] == 'TAMIL NADU'])

X = None; y = None
for i in range(tamilnadu.shape[1]-3):
    if X is None:
        X = tamilnadu[:, i:i+3]
        y = tamilnadu[:, i+3]
    else:
        X = np.concatenate((X, tamilnadu[:, i:i+3]), axis=0)
        y = np.concatenate((y, tamilnadu[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, random_state=
42)
```

In [26]:

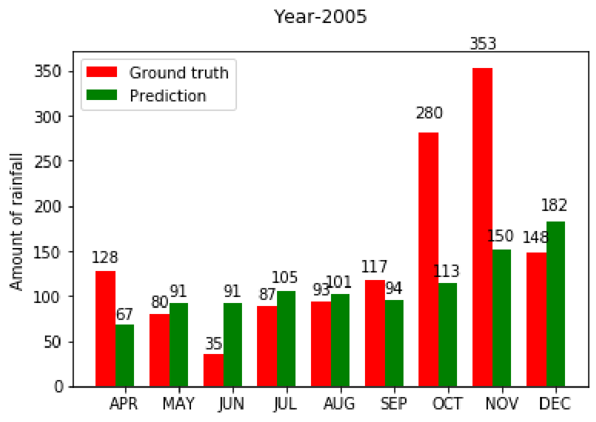
```
from sklearn import linear_model

# linear model
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print(mean_absolute_error(y_test, y_pred))
```

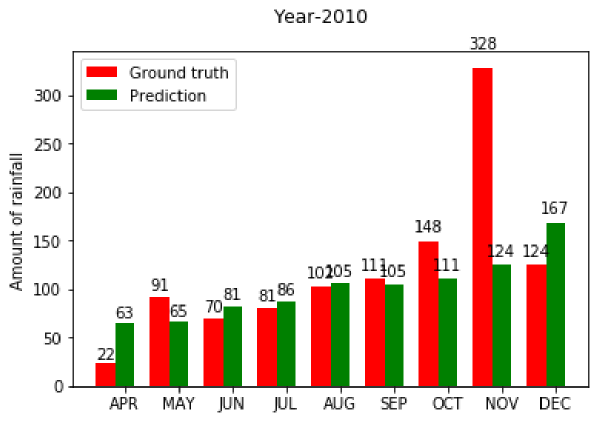
```
37.419853700246996
```

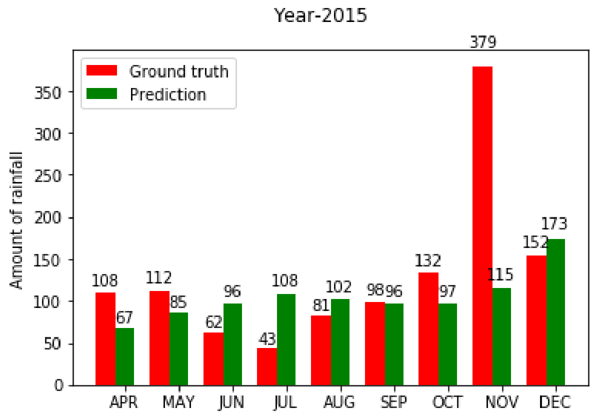
In [27]:

```
#2005
y_year_pred_2005 = reg.predict(X_year_2005)

#2010
y_year_pred_2010 = reg.predict(X_year_2010)

#2015
y_year_pred_2015 = reg.predict(X_year_2015)

print("MEAN 2005")
print(np.mean(y_year_2005),np.mean(y_year_pred_2005))
print("Standard deviation 2005")
print(np.sqrt(np.var(y_year_2005)),np.sqrt(np.var(y_year_pred_2005)))


print("MEAN 2010")
print(np.mean(y_year_2010),np.mean(y_year_pred_2010))
print("Standard deviation 2010")
print(np.sqrt(np.var(y_year_2010)),np.sqrt(np.var(y_year_pred_2010)))


print("MEAN 2015")
print(np.mean(y_year_2015),np.mean(y_year_pred_2015))
print("Standard deviation 2015")
print(np.sqrt(np.var(y_year_2015)),np.sqrt(np.var(y_year_pred_2015)))

plot_graphs(y_year_2005,y_year_pred_2005,"Year-2005")
plot_graphs(y_year_2010,y_year_pred_2010,"Year-2010")
plot_graphs(y_year_2015,y_year_pred_2015,"Year-2015")
```

```
MEAN 2005
147.3111111111111 111.15287168624565
Standard deviation 2005
97.07130763223547 32.87540585923575
MEAN 2010
120.15555555555557 101.35072505381403
Standard deviation 2010
80.8999397213223 30.369182563251652
MEAN 2015
130.25555555555556 104.79014811753537
Standard deviation 2015
93.7362625325986 27.63138542089741
```







```
In [28]:
```

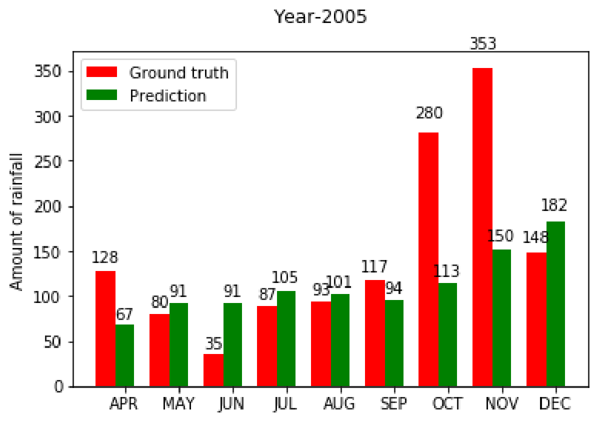
```
from sklearn.svm import SVR

# SVM model
clf = SVR(kernel='rbf', gamma='auto', C=0.5, epsilon=0.2)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(mean_absolute_error(y_test, y_pred))
```

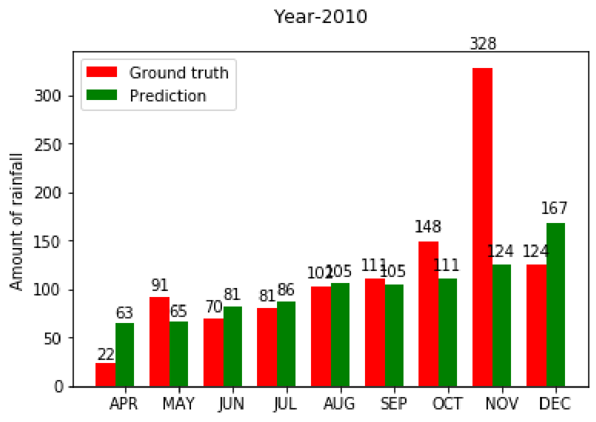
```
39.08083930391495
```

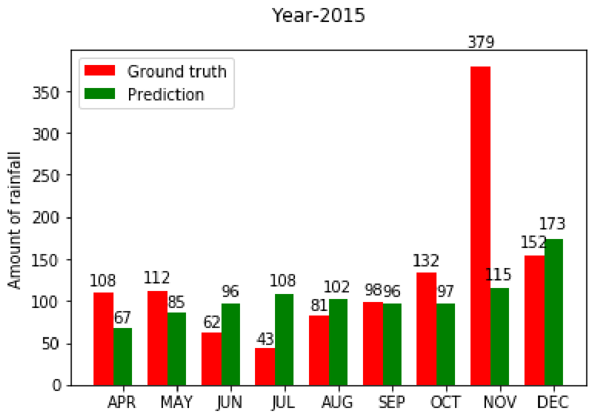
In [29]:

```
#2005
y_year_pred_2005 = reg.predict(X_year_2005)

#2010
y_year_pred_2010 = reg.predict(X_year_2010)

#2015
y_year_pred_2015 = reg.predict(X_year_2015)

print("MEAN 2005")
print(np.mean(y_year_2005),np.mean(y_year_pred_2005))
print("Standard deviation 2005")
print(np.sqrt(np.var(y_year_2005)),np.sqrt(np.var(y_year_pred_2005)))


print("MEAN 2010")
print(np.mean(y_year_2010),np.mean(y_year_pred_2010))
print("Standard deviation 2010")
print(np.sqrt(np.var(y_year_2010)),np.sqrt(np.var(y_year_pred_2010)))


print("MEAN 2015")
print(np.mean(y_year_2015),np.mean(y_year_pred_2015))
print("Standard deviation 2015")
print(np.sqrt(np.var(y_year_2015)),np.sqrt(np.var(y_year_pred_2015)))

plot_graphs(y_year_2005,y_year_pred_2005,"Year-2005")
plot_graphs(y_year_2010,y_year_pred_2010,"Year-2010")
plot_graphs(y_year_2015,y_year_pred_2015,"Year-2015")
```

```
MEAN 2005
147.3111111111111 111.15287168624565
Standard deviation 2005
97.07130763223547 32.87540585923575
MEAN 2010
120.15555555555557 101.35072505381403
Standard deviation 2010
80.8999397213223 30.369182563251652
MEAN 2015
130.25555555555556 104.79014811753537
Standard deviation 2015
93.7362625325986 27.63138542089741
```






Year-2015
Ground truth
Prediction
Amount of rainfall
0
50
100
150
200
250
300
350
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
108
67
112
85
62
96
43
108
81
102
98
96
132
97
379
115
152
173

In [30]:

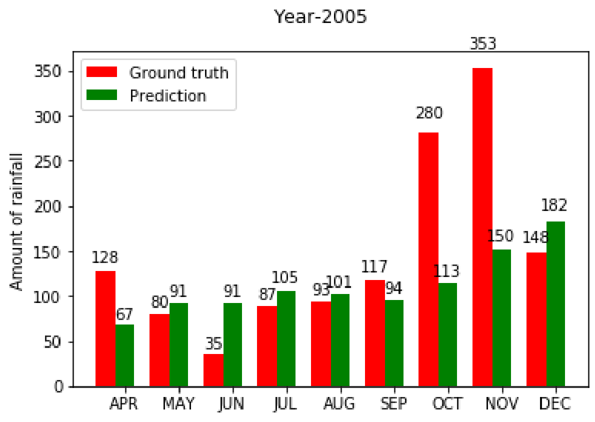
```
model.fit(x=np.expand_dims(X_train, axis=2), y=y_train, batch_size=64, epochs=10, verbo
se=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(X_test, axis=2))
print(mean_absolute_error(y_test, y_pred))
```

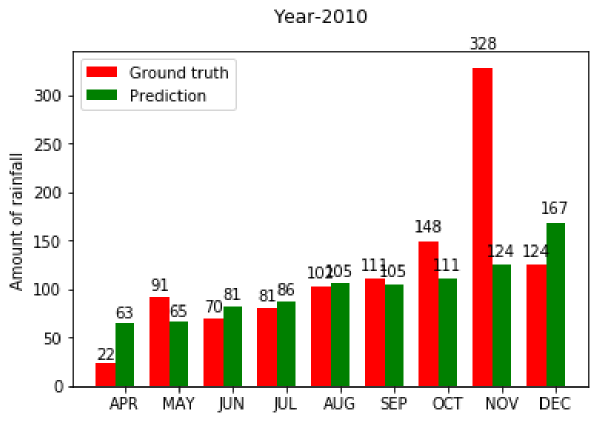
```
Train on 921 samples, validate on 103 samples
Epoch 1/10
921/921 [==============================] - 0s 201us/step - loss: 5857.5407
- mae: 59.0850 - val_loss: 5040.0638 - val_mae: 51.0546
Epoch 2/10
921/921 [==============================] - 0s 199us/step - loss: 4619.2548
- mae: 48.5622 - val_loss: 4340.1913 - val_mae: 47.9367
Epoch 3/10
921/921 [==============================] - 0s 179us/step - loss: 4098.5475
- mae: 48.2092 - val_loss: 4049.0247 - val_mae: 46.0367
Epoch 4/10
921/921 [==============================] - 0s 190us/step - loss: 3887.2704
- mae: 45.0722 - val_loss: 3918.8335 - val_mae: 44.6291
Epoch 5/10
921/921 [==============================] - 0s 180us/step - loss: 3733.3460
- mae: 45.5067 - val_loss: 3809.7137 - val_mae: 43.6893
Epoch 6/10
921/921 [==============================] - 0s 177us/step - loss: 3656.6455
- mae: 44.1687 - val_loss: 3725.6521 - val_mae: 43.2211
Epoch 7/10
921/921 [==============================] - 0s 174us/step - loss: 3637.7205
- mae: 44.8531 - val_loss: 3814.9956 - val_mae: 42.0775
Epoch 8/10
921/921 [==============================] - ETA: 0s - loss: 3687.3699 - ma
e: 43.303 - 0s 176us/step - loss: 3576.4446 - mae: 43.5117 - val_loss: 360
4.2965 - val_mae: 43.0231
Epoch 9/10
921/921 [==============================] - 0s 195us/step - loss: 3562.6583
- mae: 43.6677 - val_loss: 3592.5501 - val_mae: 41.5036
Epoch 10/10
921/921 [==============================] - 0s 178us/step - loss: 3529.0098
- mae: 43.6901 - val_loss: 3636.3927 - val_mae: 41.2031
32.8238147388805
```

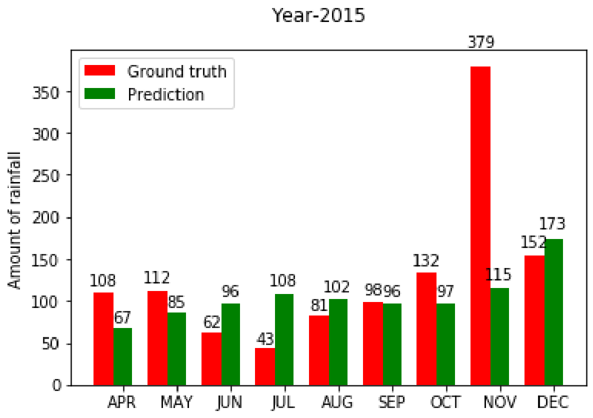
In [31]:

```
#2005
y_year_pred_2005 = reg.predict(X_year_2005)

#2010
y_year_pred_2010 = reg.predict(X_year_2010)

#2015
y_year_pred_2015 = reg.predict(X_year_2015)

print("MEAN 2005")
print(np.mean(y_year_2005),np.mean(y_year_pred_2005))
print("Standard deviation 2005")
print(np.sqrt(np.var(y_year_2005)),np.sqrt(np.var(y_year_pred_2005)))


print("MEAN 2010")
print(np.mean(y_year_2010),np.mean(y_year_pred_2010))
print("Standard deviation 2010")
print(np.sqrt(np.var(y_year_2010)),np.sqrt(np.var(y_year_pred_2010)))


print("MEAN 2015")
print(np.mean(y_year_2015),np.mean(y_year_pred_2015))
print("Standard deviation 2015")
print(np.sqrt(np.var(y_year_2015)),np.sqrt(np.var(y_year_pred_2015)))

plot_graphs(y_year_2005,y_year_pred_2005,"Year-2005")
plot_graphs(y_year_2010,y_year_pred_2010,"Year-2010")
plot_graphs(y_year_2015,y_year_pred_2015,"Year-2015")
```

```
MEAN 2005
147.3111111111111 111.15287168624565
Standard deviation 2005
97.07130763223547 32.87540585923575
MEAN 2010
120.15555555555557 101.35072505381403
Standard deviation 2010
80.8999397213223 30.369182563251652
MEAN 2015
130.25555555555556 104.79014811753537
Standard deviation 2015
93.7362625325986 27.63138542089741
```






Year-2015
Ground truth
Prediction
Amount of rainfall
0
50
100
150
200
250
300
350
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
108
67
112
85
62
96
43
108
81
102
98
96
132
97
379
115
152
173

Chapter 7

Algorithm Comparison Results

CHAPTER 8

ALGORITHM COMPARISON RESULT

7.1 Training on Complete Dataset

Algorithm	MAE
Linear Regression	96.32435229744084
SVR	127.1600615632603
Artificial Neural Nets	86.64192137579357

Table 1: Training on Complete Dataset

7.2 Training on Tamil Nadu Dataset

Algorithm	MAE
Linear Regression	37.419853700246996
SVR	39.08083930391495
Artificial Neural Nets	32.8238147388805

Table 2: Training on Tamil Nadu Dataset

- Neural Networks performs better than SVR etc.
- Bad performance by SVR model.
- Observed MAE is very high which indicates machine learning models won't work well for prediction of rainfall.
- Analysed individual year rainfall patterns for 2005, 2010, 2015.
- Approximately close Means, noticed Less Standard Deviations.

Chapter 8

Conclusion

CHAPTER 8
CONCLUSION

Various visualizations of data are observed which helps in implementing the approaches for prediction.

Various machine learning approaches such as linear regression, SVR, Artificial Neural Nets that are used to prediction.

Observations indicates machine learning models won't work well for prediction of rainfall due to fluctuations in rainfall.

Chapter 9

References

Chapter 9
References

[1] Lunagaria, M.M., Mishra, S.K. and Pandey, V. (2009). “Verification and usability of medium range weather forecast for Anand region.” J. of Agromet (Special issue): Vol. 11: 228-233.

[2] Comparative study among different neural net learning algorithms applied to rainfall time series
Surajit Chattopadhyaya and Goutami Chattopadhyay

[3] Dawid, A. P. (1984), Statistical theory: The prequential approach (with discussion), Journal of the Royal Statistical Society Ser. A, 147, 278–292.

[4] Harry R. Glahn and Dale A. Lowry
Journal of Applied Meteorology (1962-1982)
Vol. 11, No. 8 (December 1972), pp. 1203-1211

[5] Allen and Vermon (1951)

[6] Hu, M.J.C., 1964. Application of the adaline system to weather forecasting. Master Thesis, Technical Report 6775-1

[7] Journal of Environmental Management
Volume 48, Issue 1, September 1996, Pages 69-96 Regular Paper
Analysing Water Resources Alternatives and Handling Criteria by Multi Criterion Decision Techniques
Ertunga C.ÖzelkanLucien Duckstein

[8] Genetic Algorithm Approach to a Lumber Cutting Optimization Problem
Deborah F. Cook &Mary Leigh Wolfe

[9] Classification of Hydrometeors Based on Polarimetric Radar Measurements: Development of Fuzzy Logic and Neuro-Fuzzy Systems, and in Situ Verification
Hongping Liu and V. Chandrasekar
Colorado State University, Fort Collins, Colorado

[10] Critical constructs for analyzing e-businesses: investment, user experience and revenue models
Rahul De, Biju Mathew, Dolphy M. Abraham

[11] The Impact of Some Environmental Factors on the Fecundity of Phenacoccus Solenopsis Tinsley (Hemiptera: Pseudococcidae): A Serious Pest of Cotton And Other Crops
Ghulam Abbas, Muhammad Jalal Arif, Muhammad Ashfaq, Muhammad Aslam and Shafqat Saeed

[12] An ANN application for water quality forecasting
SundarambalPalani, Shie-Yui Liong, Pavel Tkalich

[13] Vulnerability of Korean water resources to climate change and population growth
H. Chang; J. Franczyk; E.-S. Im; W.-T. Kwon; D.-H. Bae; I.-W. Jung

[14] A novel approach for the elimination of artefacts from EEG signals employing an improved Artificial Immune System algorithm
S. Suja Priyadharsini,S. Edward Rajan &S. Femilin Sheniha

[15] Radial basis function Network based on multi-objective particle swarm optimization
Sultan Noman Qasem ; Siti Mariyam Hj. Shamsuddin

[16] Application of fuzzy sets to climatic classification
Alex B Mcbratney, A.W Moore

[17] Neurocomputing based Canadian weather analysis
Imran Maqsood, Muhammad Riaz Khan, Ajith Abraham

[18] Application of soft computing models to hourly weather analysis in southern Saskatchewan, Canada
Imran Maqsooda, Muhammad Riaz, Khan Guo , H.Huanga Rifaat Abdallac

[19] A comparative study of medium-weather-dependent load forecasting using enhanced artificial/fuzzy neural network and statistical techniques
M.M.Elkateb, Khalid Solaimana, YusufAl-Turkib

[20] A Knowledge-Based Theory of Inter-Firm Collaboration
Robert M. Grant, Charles Baden-Fuller

[21] Annual Rainfall Forecasting by Using Mamdani Fuzzy Inference System
Gholam Abbas Fallah-Ghalhary
Mohammad Mousavi-Baygi and Majid Habibi Nokhandan

Gholam Abbas (2009), Maqsood et al. (2004), Paras et al. (2008), Singh et al. (2006), Abraham et al. (2004), uses temperature, relative humidity and vapour pressure

[22] Prediction of thermal conductivity of granite rocks from porosity and density data at normal temperature and pressure: in situ thermal conductivity measurements
Asghari Maqsood, Kashif Kamran and Iftikhar Hussain Gul

[23] Experimental study of bubble formation at metal porous spargers: Effect of liquid properties and sparger characteristics on the initial bubble size distribution
N.A.Kazakis, A.A.Mouza, S.V.Paras

[24] System and method for load balancing virtual machines in a computer network
Sumankumar Singh, Timothy Abels

[25] A compact wireless gas sensor using a carbon nanotube/PMMA thin film chemiresistor Jose K Abraham1, Biju Philip, Ashwin Witchurch, Vijay K Varadan and C Channa Reddy