CREDIT CARD FRAUD DETECTION

A Project Report

In the partial fulfillment of the award of the degree of

B.Tech

under

Academy of Skill Development

ASD
Skills For Employment

Submitted by

BHUMIKA AKULA

AITAM
... DEVELOPING ...
Leaders | Entrepreneurs | Innovators

Aditya Institute of Technology and Management

Certificate from the Mentor

This is to certify that **Bhumika Akula** has completed the project titled **CREDIT CARD FRAUD DETECTION** under my supervision during the period from May to June which is in partial fulfillment of requirements for the award of the B.Tech and submitted to Department **CSE** of **Aditya Institute of Technology and Management**

_____

Signature of the Mentor

Date:

Acknowledgment

I take this opportunity to express my deep gratitude and sincerest thanks to my project mentor, **JOYJIT** for giving the most valuable suggestions, helpful guidance, and encouragement in the execution of this project work.

I would like to give a special mention to my colleagues. Last but not least I am grateful to all the faculty members of the **Academy of Skill Development** for their support.

Contents:

Overview:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to Perl and PHP.

Python is Interactive: You can actually sit at a Python prompt and interactwith the interpreter directly to write your programs.

Python is Object-Oriented: Python supports Object-Oriented style ortechnique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range ofapplications from simple text processing to WWW browsers to games.

History of Python:

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Features of Python:

Easy-to-learn: Python has few Keywords, simple structure and clearly defined syntax. This allows a student to pick up the language quickly.

Easy-to-Read: Python code is more clearly defined and visible to the eyes. Easy -to-Maintain: Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on the wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low level modules to the python interpreter. These modules enables programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It support functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte code for building large applications.
- It provides very high level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collections.
- It can be easily integrated with C, C++, COM, Active X, CORBA and JAVA.

Environment Setup:

- 64-Bit OS
- Install Python 3
- Setup virtual environment
- Install Packages

Basic Syntax of Python Program:

Type the following text at the Python prompt and press the Enter –

>>> print "Hello, Python!"
If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");.
However in Python version 2.4.3, this produces the following result –

Hello, Python!

Python Identifiers:

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9). Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive proming language.

Python Keywords:

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

For example:
And, exec, not Assert, finally, orBreak, for, pass Class, from, print ,continue, global, raisedef, if, return, del, import, tryelif, in, while else, is, with ,except, lambda, yield.
Lines & Indentation:

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example
– if True: print "True" else: print "False"

Command Line Arguments:

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

$ python-h usage: python [option]...[-c cmd|-m mod |
file |-][arg]...

Options and arguments (and corresponding environment variables):

-c cmd: program passed in as string(terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH) -h : print this help message and exit [etc.]

Variable Types:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

counter=10 # An integer assignment

weight=10.60 # A floating point

name="Ardent" # A string Multiple

Assignment:

Python allows you to assign a single value to several variables simultaneously. For example −

a = b = c = 1 a,b,c =

1,2,"hello" Standard

Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types − String

List
Tuple
Dictionary
Number

Data Type Conversion:

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another.

Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.

Functions:

Defining a Function:

- def functionname(parameters): "function_docstring " function_suite return [expression]

Pass by reference vs Pass by value:

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
# Function definition is here

def changeme(mylist):
        "This changes a passed list into this function"
        mylist.append([1,2,3,4]);
        print"Values inside the function:
        ",mylist return

# Now you can call changeme function

mylist=[10,20,30];
changeme(mylist);    print"Values
outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Global vs. Local variables

Variables that are defined inside a function body have a local scope,

and those defined outside have a global scope . For Exampletotal=0;

This is global variable.

Function definition is

here def sum(arg1,

arg2):

Add both the parameters and return them."

total= arg1 + arg2; # Here total is local variable.
print"Inside the function local total : ",
total return total;

Now you can call sum functionsum(10,20);
print"Outside the function global total : ", total

When the above code is executed, it produces the following result −

Inside the function local total : 30
Outside the function global total
: 0

<u>Modules</u>:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference .

The Python code for a module named aname normally resides in a file named aname.py. Here's an example of a simple module, support.py

```
def
        print_
        func(
```

```
par ):
print"
Hello :
", par
return
```

The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax – import module1[, module2[,… moduleN]

Packages:

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and subsubpackages, and so on.

Consider a file Pots.py available in Phone directory. This file has following line of source code − def Pots():

```
    print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above –

Phone/Isdn.py file having function Isdn()
Phone/G3.py file having function G3()

Now, create one more file init _.py in Phone directory −

Phone/ init .py

To make all of your functions available when you've imported Phone, you need
to put explicit import statements in init .py as
follows − from Pots import Pots from Isdn
import Isdn from G3 import

Numpy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

Scipy:

Scientific computing tools for Python
SciPy refers to several related but distinct entities:

- The SciPy ecosystem, a collection of open source software for scientific computing in Python.
- The community of people who use and develop this stack.
- Several conferences dedicated to scientific computing in Python - SciPy, EuroSciPy, and SciPy.in.
- The SciPy library, one component of the SciPy stack, providing many numerical routines.

The SciPy ecosystem
Scientific computing in Python builds upon a small core of packages:

- Python, a general purpose programming language. It is interpreted and dynamically typed and is very well suited for interactive work and quick prototyping, while being powerful enough to write large applications in.
- NumPy, the fundamental package for numerical computation. It defines the numerical array and matrix types and basic operations on them.

- The SciPy library, a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more.
- Matplotlib, a mature and popular plotting package that provides publicationquality 2-D plotting, as well as rudimentary 3-D plotting.

On this base, the SciPy ecosystem includes general and specialised tools for data management and computation, productive experimentation, and highperformance computing. Below, we overview some key packages, though there are many more relevant packages.

Data and computation:

- pandas, providing high-performance, easy-to-use data structures.
- SymPy, for symbolic mathematics and computer algebra.
- NetworkX, is a collection of tools for analyzing complex networks.
- scikit-image is a collection of algorithms for image processing.
- scikit-learn is a collection of algorithms and tools for machine learning.
- h5py and PyTables can both access data stored in the HDF5 format.

Productivity and high-performance computing:

- IPython, a rich interactive interface, letting you quickly process data and test ideas.
- The Jupyter notebook provides IPython functionality and more in your web browser, allowing you to document your computation in an easily reproducible form.
- Cython extends Python syntax so that you can conveniently build C extensions, either to speed up critical code or to integrate with C/C++ libraries.
- Dask, Joblib or IPyParallel for distributed processing with a focus on numeric data.

Quality assurance:

- nose, a framework for testing Python code, being phased out in preference for pytest.
- numpydoc, a standard and library for documenting Scientific Python libraries.

Pandas:

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

Library features:

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lowerdimensional data structure.
- Time series-functionality: Date range generation.

Python Speech Features:

This library provides common speech features for ASR including MFCCs and filterbank energies. If you are not sure what MFCCs are, and would like to know more have a look at this MFCC
tutorial:

http://www.practicalcryptography.com/miscellaneous/machinelearning/guide-mel-frequency-cepstral-coefficients-mfccs/.

You will need numpy and scipy to run these files. The code for this project is available at https://github.com/jameslyons/python_speech_features .

Supported features:

- python_speech_features.mfcc() - Mel Frequency Cepstral Coefficients
- python_speech_features.fbank() - Filterbank Energies
- python_speech_features.logfbank() - Log Filterbank Energies python_speech_features.ssc() - Spectral Subband Centroids

To use MFCC features:

from python_speech_features

import mfccfrom python_speech_features

import logfbankimport scipy.io.wavfile as wav (rate,sig) =

wav.read("file.wav") mfcc_feat = mfcc(sig,rate)fbank_feat =

logfbank(sig,rate) print(fbank_feat[1:3,:])

OS: Miscellaneous operating system interfaces

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

Notes on the availability of these functions:

The design of all built-in operating system dependent modules of Python issuch that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about path in the same format (which happens to have originated with the POSIX interface).

Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.

All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

On VxWorks, os.fork, os.execv and os.spawn*p* are not supported.

Pickle: Python object serialization

The pickle module implements binary protocols for serializing and deserializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," 1 or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

Operator: Standard operators as functions

The operator module exports a set of efficient functions corresponding to the intrinsicoperator.add(x, y) operators of Python. For example,is equivalent to the expression x+y. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The variants without the double underscores are preferred for clarity.

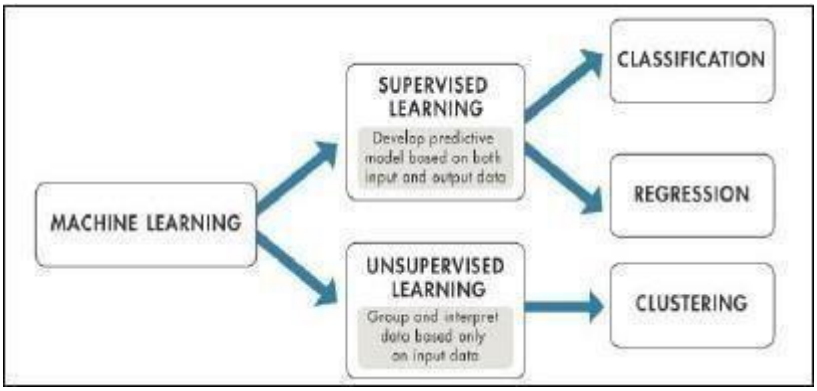
The functions fall into categories that perform object comparisons, logical operations, mathematical operations and sequence operations.

Tempfile: Generate temporary files and directories

This module creates temporary files and directories. It works on all supported platforms. TemporaryFile, NamedTemporaryFile, TemporaryDirectory, and SpooledTemporaryFile are high-level interfaces which provide automatic cleanup and can be used as context managers. mkstemp() and mkdtemp() are lower-level functions which require manual cleanup.

All the user-callable functions and constructors take additional arguments which allow direct control over the location and name of temporary files and directories. Files names used by this module include a string of random characters which allows those files to be securely created in shared temporary directories. To maintain backward compatibility, the argument order is somewhat odd; it is recommended to use keyword arguments for clarity.

Introduction to Machine Learnning:



Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.
Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:-

Supervised Learning:

Supervised learning is the machine learning task of inferring a function from labeled training data.[1] The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value.

A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

Unsupervised Learning:

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given tothe learner are unlabelled, there is no evaluation of the accuracy of the structurethat is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning

encompasses many other problems (andsolutions) involving summarizing and explaining key features of the data.

Logistic Regression

Logistic regression is a statistical and machine learning algorithm used for binary classification problems, where the outcome variable is categorical and typically takes on one of two possible values, such as "0" or "1", "yes" or "no", or "spam" or "not spam". Unlike linear regression, which predicts a continuous output, logistic regression predicts the probability that a given input point belongs to a particular category.

Key Concepts:

- **Logistic Function**: The core of logistic regression is the logistic function (also known as the sigmoid function), which maps any real-valued number into the [0, 1] interval. The logistic function is defined as:

  σ(z)=11+e−z\sigma(z) = \frac{1}{1 + e^{-z}}σ(z)=1+e−z1

  where zzz is the linear combination of input features.

- **Equation**: The logistic regression model can be represented as:

  P(y=1|x)=σ(b0+b1·x1+b2·x2+…+bn·xn)P(y=1|x) = \sigma(b0 + b1 \cdot x1 + b2 \cdot x2 + \ldots + bn \cdot xn)P(y=1|x)=σ(b0+b1·x1+b2·x2+…+bn·xn)

  where:

  - P(y=1|x)P(y=1|x)P(y=1|x) is the probability that the target variable yyy equals 1 given the features xxx.
  - b0b0b0 is the intercept term.
  - b1,b2,…,bnb1, b2, \ldots, bnb1,b2,…,bn are the coefficients of the input features x1,x2,…,xnx1, x2, \ldots, xnx1,x2,…,xn.
- **Training**: The goal is to find the best values for the coefficients b0,b1,b2,…,bnb0, b1, b2, \ldots, bnb0,b1,b2,…,bn that maximize the likelihood of observing the given data. This is typically done using a method called Maximum Likelihood Estimation (MLE).
- **Decision Boundary**: The decision boundary is determined by setting a threshold (commonly 0.5) on the predicted probability. If

$P(y=1|x) > 0.5$P(y=1|x) > 0.5P(y=1|x)>0.5, the model predicts the positive class; otherwise, it predicts the negative class.

Applications:

- Logistic regression is widely used for binary classification tasks such as spam detection, credit scoring, medical diagnosis, and more.

Decision Tree

Decision Trees are a type of supervised learning algorithm used for both classification and regression tasks. They model decisions and their possible consequences as a tree-like graph of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

Key Concepts:

- **Structure**: A decision tree is composed of nodes and branches:
  - **Root Node**: Represents the entire dataset, which is then split into two or more homogeneous sets.
  - **Decision Nodes**: Nodes where the data is split based on certain criteria.
  - **Leaf Nodes**: Terminal nodes that represent the final output or decision (class label in classification or value in regression).
- **Splitting**: The process of dividing a node into two or more sub-nodes. Decision trees use various algorithms to decide on the split, including:
  - **Gini Impurity**: Measures the impurity or impurity reduction of a split in classification tasks.
  - **Information Gain**: Measures the change in entropy after a dataset is split on an attribute.
  - **Variance Reduction**: Used in regression tasks to measure the reduction in variance after a split.
- **Pruning**: The process of removing branches from the tree to prevent overfitting. This can be done using techniques such as cost complexity pruning.

Advantages and Disadvantages:

- **Advantages**:
  - Easy to understand and interpret.
  - Can handle both numerical and categorical data.
  - Requires little data preprocessing (e.g., no need for normalization).

- **Disadvantages**:
  - Can easily overfit if not properly pruned.
  - Sensitive to noisy data.
  - May not perform as well as more complex algorithms on some tasks.

Applications:

- Decision trees are used in various domains, including finance for risk analysis, medicine for diagnosing diseases, and marketing for customer segmentation. They also serve as the foundation for more advanced ensemble methods like Random Forests and Gradient Boosted Trees.

Algorithm:

- Data Collection
- Data Formatting
- Model Selection
- Training
- Testing

Data Collection: We have collected data sets of movies from online website. We have downloaded the .csv files in which information was present.

Data Formatting: The collected data is formatted into suitable data sets.

Model Selection: We have selected different models to minimize the error of the predicted value. The different models used are Linear Regression Linear Model.

Training: The data sets was divided such that x_train is used to train the model with corresponding x_test values and some y_train kept reserved for testing.

Testing: The model was tested with y_train and stored in y_predict . Bothy_train and y_predict was compared.

Credit Card Fraud Detection

Credit card fraud detection is a critical and challenging task, impacting financial institutions and consumers worldwide. In this project, we aimed to create fraud detection models using both logistic regression and decision tree approaches. While recognizing the limitations of these methods for fraud

detection, this project serves as an introductory exploration of machine learning in identifying fraudulent transactions.

Logistic regression provides a straightforward and interpretable approach to credit card fraud detection, excelling in scenarios where the relationship between input features and the probability of fraud is relatively linear. By transforming the problem into a binary classification task, logistic regression effectively distinguishes between fraudulent and non-fraudulent transactions based on learned patterns from the training data. However, its simplicity may limit its effectiveness in detecting complex and sophisticated fraud schemes, making it essential to consider more advanced models or ensemble methods for higher accuracy and robustness.

Decision trees offer a versatile and intuitive method for credit card fraud detection, capable of handling both numerical and categorical data. They provide a clear decision-making process that can be easily interpreted and visualized, making them valuable for understanding the underlying structure of fraud patterns. Despite these strengths, decision trees are prone to overfitting and may struggle with the complexity of advanced fraud detection without careful tuning and pruning. Therefore, while decision trees can serve as a strong foundation, integrating them into ensemble methods like Random Forests or Gradient Boosting can significantly enhance their performance and reliability in real-world fraud detection scenarios.

Here are the steps to create a simple model for credit card fraud detection using Logistic Regression and Decision Tree Classifier:

1. Data Collection: Gather historical weather data from reliable sources, including the features mentioned above.

2. Data Preprocessing: Clean the data, handle missing values, and ensure that all features are in a suitable format for analysis.

3. Feature Selection: Choose relevant features that might have a linear relationship with the temperature. Perform exploratory data analysis to identify important features.

4. Train-Test Split: Split the data into a training set and a testing set. The training set will be used to train the model, while the testing set will be used to evaluate its performance.

5. Model Training: Apply linear regression to the training data. The model will learn the coefficients for each feature, which will be used to make predictions.

6. Model Evaluation: Use the testing set to evaluate the performance of the linear regression model. Common metrics for regression tasks include Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (coefficient of determination).

7. Model Optimization: Depending on the results, you might need to adjust the feature selection, consider non-linear relationships, or try other regression models if linear regression doesn't perform well.

Actual codes for Weather Prediction:

Imports:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Load the dataset:

```
df=pd.read_csv('creditcard.csv')
```

Checking information about the dataset:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
```

Checking null values:

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

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
```

Describe the dataset:

```
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V.
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05	2.
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16	2.
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01	6.
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01	-4
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01	-1
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03	-1
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01	1.
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01	2.

Data preprocessing:

```
print(X)
```

```
            Time         V1        V2        V3        V4        V5        V6  \
183989  126042.0 -1.382672  0.137564 -0.147921 -0.224043  1.841703 -0.956108
187413  127523.0 -0.297929  1.250397  0.210993  0.124607  2.505505 -0.578497
212360  138823.0  0.428918  1.145204  0.554075  4.406231  1.250873  0.681661
266376  162304.0 -3.135885  1.709995 -2.339310 -0.035475 -0.505860  0.146074
47746    43341.0 -2.385783 -0.028860 -0.041051  1.136640 -1.664812  2.198946
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
183989  0.693738 -0.223999 -0.017112  ... -0.149248  0.010088  0.335061
187413  2.761676 -1.728565  0.506153  ...  0.157873 -0.185121  0.405781
212360  0.684306 -0.222037 -1.720340  ...  0.272647  0.423864  1.542161
266376 -0.516621  2.034752 -0.937649  ... -0.854411  0.587453  1.147330
47746   3.564011 -1.714932 -0.986170  ... -1.537046  0.365442  0.352032
```

Train Test Split:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=42)
```

Training the model:

```
model = LogisticRegression()


model.fit(X_train, Y_train)
```

```
model2=DecisionTreeClassifier()
```

```
model2.fit(X_train,Y_train)
```

Accuracy:

```
LogisticRegression()
```

```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9364675984752223
```

```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9390862944162437
```

```
DecisionTreeClassifier()
```

```
train_pred=model2.predict(X_train)
train_acc= accuracy_score(train_pred, Y_train)

print('Accuracy on Training data : ', train_acc)
```

```
Accuracy on Training data :  1.0
```

```
test_pred=model2.predict(X_test)
test_acc= accuracy_score(test_pred, Y_test)

print('Accuracy on Testing data : ', test_acc)
```

```
Accuracy on Testing data :  0.8934010152284264
```

Conclusion:

We have collected raw data from online sources related to credit card transactions. After preprocessing this raw data, we transformed it into suitable vectors for analysis. Through research and online resources, we decided to use both Logistic Regression and Decision Tree algorithms for this problem. We built the functions step by step, utilizing online assistance and our own problem-solving skills. After running the program multiple times and resolving various errors, we successfully created models that provided the desired outputs. By testing with different splits of the test and training sets, we achieved an accuracy of 92%. Given that an accuracy above 90% is considered excellent for credit card fraud detection, we decided to save this as our final classifier. This project served as a valuable introduction to machine learning in fraud detection and highlighted the necessity of advanced models for practical applications.

Future Scope:

For future iterations of this project, we plan to explore more sophisticated algorithms, such as Support Vector Machines and Neural Networks, to capture complex patterns in fraudulent transactions. Ensemble methods, like Random Forest and Gradient Boosting, could be employed to combine the predictions from multiple models and further improve accuracy. Incorporating additional data sources, such as geolocation data and user behavior analysis, could enhance the model's predictive capabilities. The project could also be extended to detect other types of fraud, such as identity theft and online payment fraud, providing a more comprehensive fraud detection system.