RAMNIRANJAN JHUNJHUNWALA COLLEGE GHATKOPAR (W), MUMBAI - 400 086

DEPARTMENT OF INFORMATION TECHNOLOGY 2020 - 2021

M.Sc. (I.T.) SEM II

Advanced Artificial Intelligence

Name : Sneha Ramchandra Pawar.
Roll No. : 11

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that Miss. **Sneha Ramchandra Pawar.** with Roll No. **11** has successfully completed the necessary course of experiments in the subject of **Advanced Artificial Intelligence** during the academic year **2020 – 2021** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc.(IT)** semester -II.

_____ _____

Internal Examiner External Examiner

_____ _____

Head of Department College Seal

INDEX

Practical No.	Practical Title	Date
1	Heuristic Search - Hill Climbing - A* algorithm	10/07/21
2	Reinforcement Learning: Markov decision	19/07/21
3	Reinforcement Learning: Monte Carlo Prediction	19/07/21
4	Predictive Analytics - Forecasting (Logistic, Time Series - ARIMA, Case Study)	24/07/21
5	Ensemble Techniques (Boosting, Bagging)	19/07/21
6	Prolog - Color mapping, Cryptarthmatic problem	24/07/21
7	Travelling Salesman Problem	10/07/21
8	Checkerboard	24/07/21

PRACTICAL – 1

Heuristic Search Techniques

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed.

A Heuristic (or a heuristic function) takes a look at search algorithms. At each branching step, it evaluates the available information and makes a decision on which branch to follow. It does so by ranking alternatives.

Briefly, we can taxonomize such techniques of Heuristic into two categories:

a. Direct Heuristic Search Techniques

Other names for these are Blind Search, Uninformed Search, and Blind Control Strategy. They search the entire state space for a solution and use an arbitrary ordering of operations. Examples of these are Breadth First Search (BFS) and Depth First Search (DFS).

b. Weak Heuristic Search Techniques

Other names for these are Informed Search, Heuristic Search, and Heuristic Control Strategy. These are effective if applied correctly to the right types of tasks and usually demand domain-specific information. We need this extra information to compute preference among child nodes to explore and expand.

Examples are Best First Search (BFS), A* Search, Bidirectional Search, Tabu Search, Beam Search, Simulated Annealing, Hill Climbing & Constraint Satisfaction Problems

Hill Climbing in Artifical Intelligence

This is a heuristic for optimizing problems mathematically. We need to choose values from the input to maximize or minimize a real function

• It is a variant of the generate-and-test algorithm

• It makes use of the greedy approach

This means it keeps generating possible solutions until it finds the expected solution, and moves only in the direction which optimizes the cost function for it.

Types of Hill Climbing

• **Simple Hill Climbing-** This examines one neighboring node at a time and selects the first one that optimizes the current cost to be the next node.

• **Steepest Ascent Hill Climbing-** This examines all neighboring nodes and

selects the one closest to the solution state.

• **Stochastic Hill Climbing-** This selects a neighboring node at random and decides whether to move to it or examine another.

Let's take a look at the algorithm for simple hill climbing.

1. Evaluate initial state- if goal state, stop and return success. Else, make initial state current.

2. Loop until the solution reached or until no new operators left to apply to current state:

a. Select new operator to apply to the current producing new state.

b. Evaluate new state:

• If a goal state, stop and return success.

• If better than the current state, make it current state, proceed.

• Even if not better than the current state, continue until the solution reached.

3. Exit.

Problems with Hill Climbing

• **Local Maximum-** All neighboring states have values worse than the current. The greedy approach means we won't be moving to a worse state. This terminates the process even though there may have been a better solution. As a workaround, we use backtracking.

• **Plateau-** All neighbors to it have the same value. This makes it impossible to choose a direction. To avoid this, we randomly make a big jump.

• **Ridge-** At a ridge, movement in all possible directions is downward. This makes it look like a peak and terminates the process. To avoid this, we may use two or more rules before testing.

CODE:

```
import random
import string

def generate_random_solution(length=13):
    return[random.choice(string.printable) for _ in range(length)]

def evaluate(solution):
   target = list("Hello, World!")
   diff = 0
   for i in range(len(target)):
        s=solution[i]
        t=target[i]
```

```
        diff += abs(ord(s) - ord(t))
    return diff
def mutate_solution(solution):
    index = random.randint(0, len(solution) - 1)
    solution[index] = random.choice(string.printable)
best = generate_random_solution()
best_score = evaluate(best)

while True:
  print('Best score so far', best_score, 'Solution',"".join(best))
  if best_score == 0:
     break
  new_solution = list(best)
  mutate_solution(new_solution)
  score = evaluate(new_solution)
  if evaluate(new_solution) < best_score:
     best = new_solution
     best_score = score
```

OUTPUT:

```
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
```

Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 1 Solution Hello, Worlc!
Best score so far 0 Solution Hello, World!

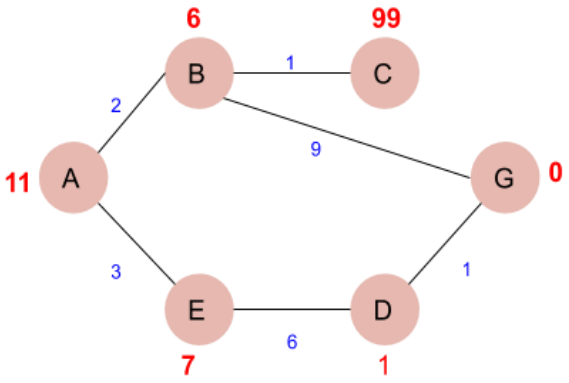
A* ALGORITHM

- A* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently.
- It is essentially a best first search algorithm.

WORKING

- It maintains a tree of paths originating at the start node.
- It extends those paths one edge at a time.
- It continues until its termination criterion is satisfied.
- A* Algorithm extends the path that minimizes the following function-
- **f(n) = g(n) + h(n)**
- g(n) = shows the shortest path's value from the starting node to node n
  h(n) = The heuristic approximation of the value of the node

PROBLEM



Step 1

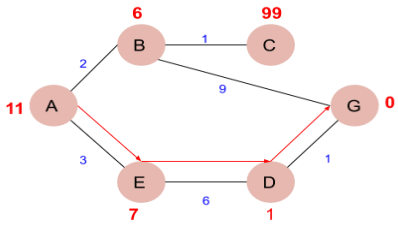
- **f(n) = g(n) + h(n)**
- A → B = 2 + 6 =8
- A → E = 3 + 6 = 9

Step 2

- A → B → C = (2 + 1) + 99= 102
- A → B → G = (2 + 9) + 0 = 11
- A → E → D = (3 + 6) + 1 = 10

Step 3

- A → E → D → G = (3 + 6 + 1) +0 =11



CODE:

```
pip install simpleai
from simpleai.search import SearchProblem, astar

GOAL = 'HELLO WORLD'
class HelloProblem(SearchProblem):
    def actions(self, state):
        if len(state) < len(GOAL):
            return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ')
        else:
            return []
    def result(self, state, action):
        return state + action
    def is_goal(self, state):
        return state == GOAL
    def heuristic(self, state):
 # how far are we from the goal?
     wrong = sum([1 if state[i] != GOAL[i] else 0
          for i in range(len(state))])
     missing = len(GOAL) - len(state)
     return wrong + missing

problem = HelloProblem(initial_state='')
result = astar(problem)
print(result.path())
print(result.state)
```

OUTPUT:

```
[(None, ''), ('H', 'H'), ('E', 'HE'), ('L', 'HEL'), ('L', 'HELL'), ('O', 'HELLO'),
(' ', 'HELLO '), ('W', 'HELLO W'), ('O', 'HELLO WO'), ('R', 'HELLO WOR'), ('L',
'HELLO WORL'), ('D', 'HELLO WORLD')]
HELLO WORLD
```

Google Colab Noteboot Link:

https://colab.research.google.com/drive/1-Hnik688MpxQVul-4nAcdc34H7ZRNFBn?usp=sharing

PRACTICAL – 2

Reinforcement Learning: Markov decision

A Markov Model is a stochastic model that models random variables in such a manner that the variables follow the Markov property. Markov chains are used in text generation and auto-completion applications. In a Markov Process, we use a matrix to represent the transition probabilities from one state to another. This matrix is called the Transition or probability matrix. It is usually denoted by P.

$$P = \begin{bmatrix} p_{11} & p_{12} & . & . & . & p_{1r} \\ p_{21} & p_{22} & . & . & . & p_{2r} \\ . & . & & . & & . \\ . & . & & . & & . \\ p_{21} & p_{22} & . & . & . & p_{2r} \end{bmatrix}$$

Note, pij≥0, and 'i' for all values is,

$$\sum_{k=1}^{r} p_{ik} = \sum_{k=1}^{r} P(X_{m+1} = k | X_m = \mathrm{i})$$

A Markov model is represented by a State Transition Diagram. The diagram shows the transitions among the different states in a Markov Chain.

Markov Chain Applications

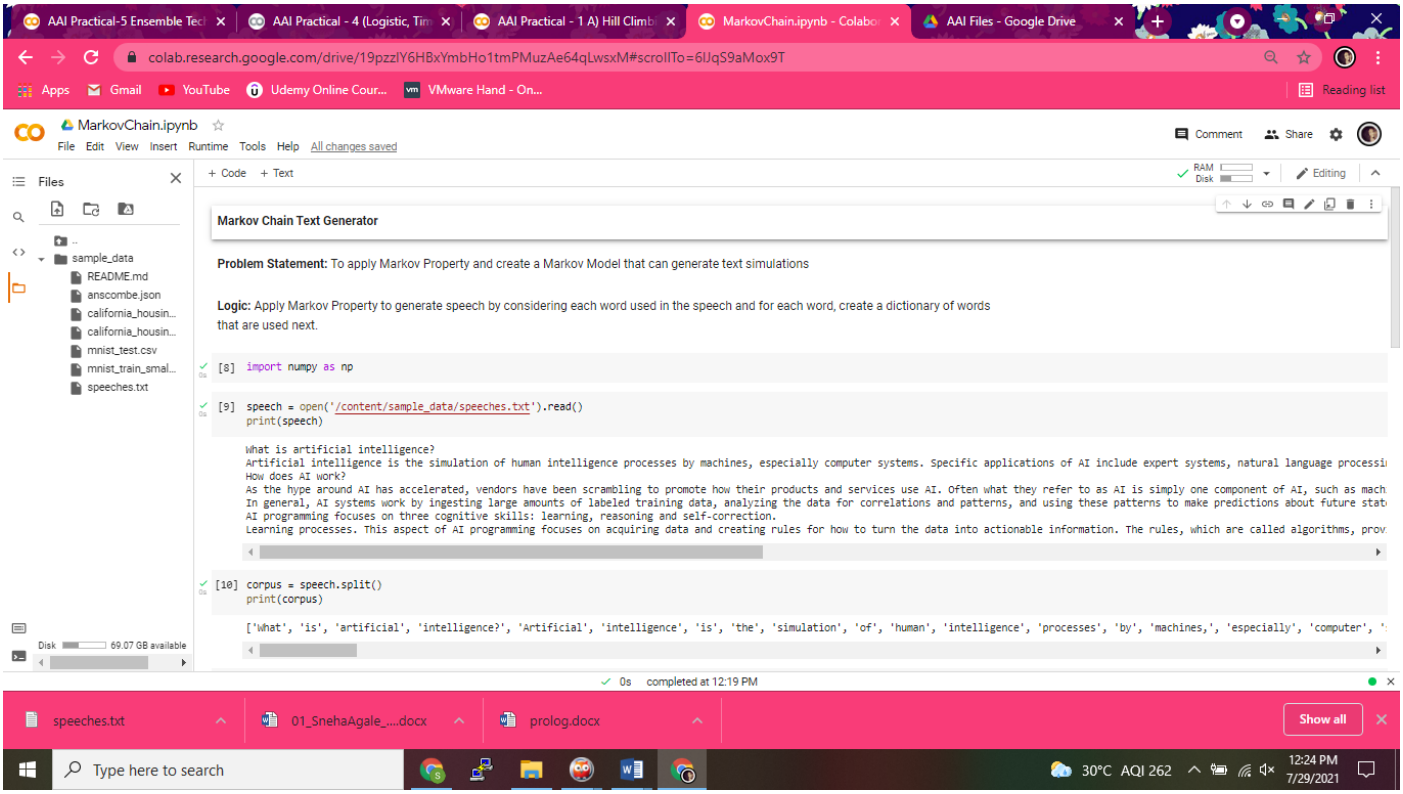
Here's a list of real-world applications of Markov chains:

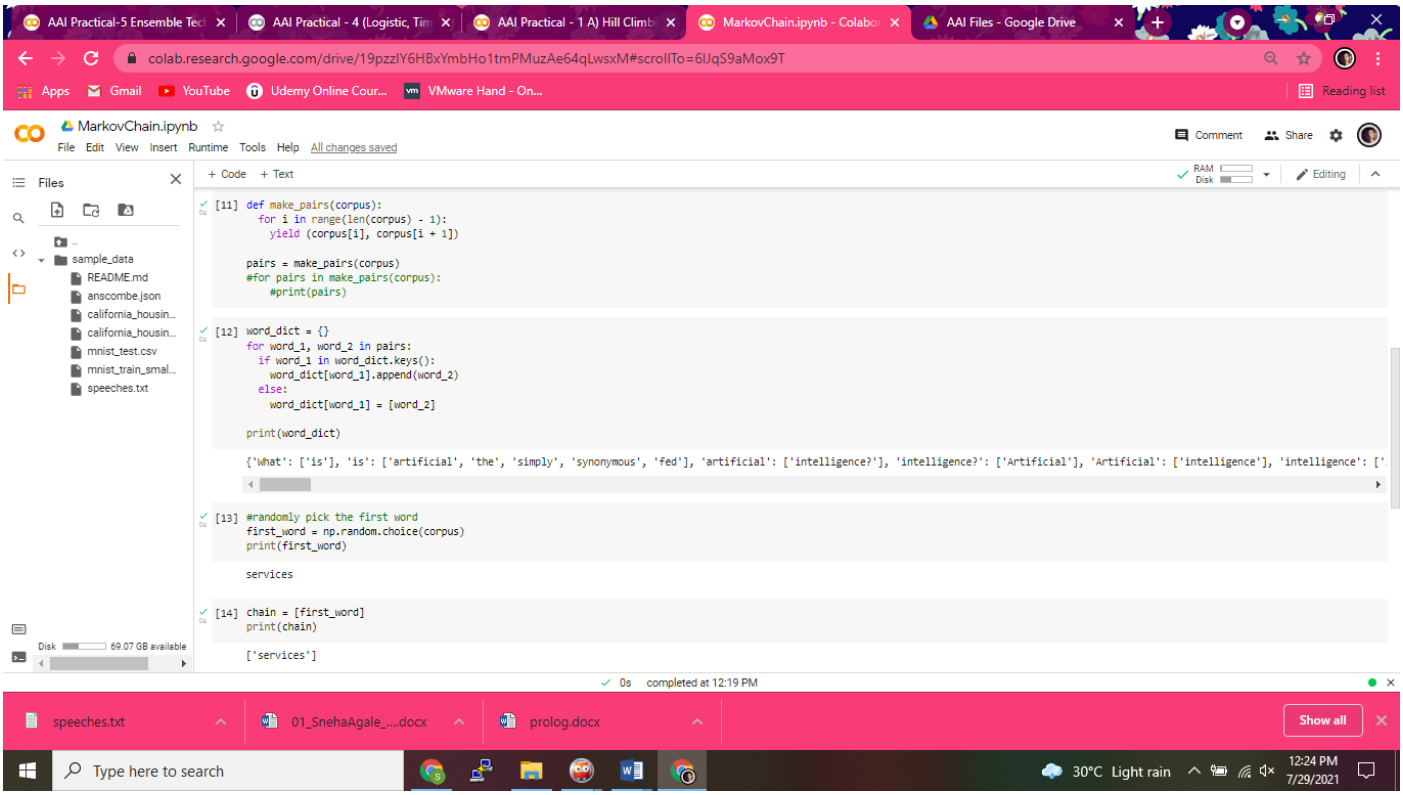
1. **Google PageRank:** The entire web can be thought of as a Markov model, where every web page can be a state and the links or references between these pages can be thought of as, transitions with probabilities. So basically, irrespective of which web page you start surfing on, the chance of getting to a certain web page, say, X is a fixed probability.
2. **Typing Word Prediction:** Markov chains are known to be used for predicting upcoming words. They can also be used in auto-completion and suggestions.
3. **Subreddit Simulation:** Surely you've come across Reddit and had an interaction on one of their threads or subreddits. Reddit uses a subreddit simulator that consumes a huge amount of data containing all the comments and discussions held across their groups. By making use of Markov chains, the simulator produces word-to-word probabilities, to create comments and topics.
4. **Text generator:** Markov chains are most commonly used to generate dummy texts or produce large essays and compile speeches. It is also used in the name generators that you see on the web.

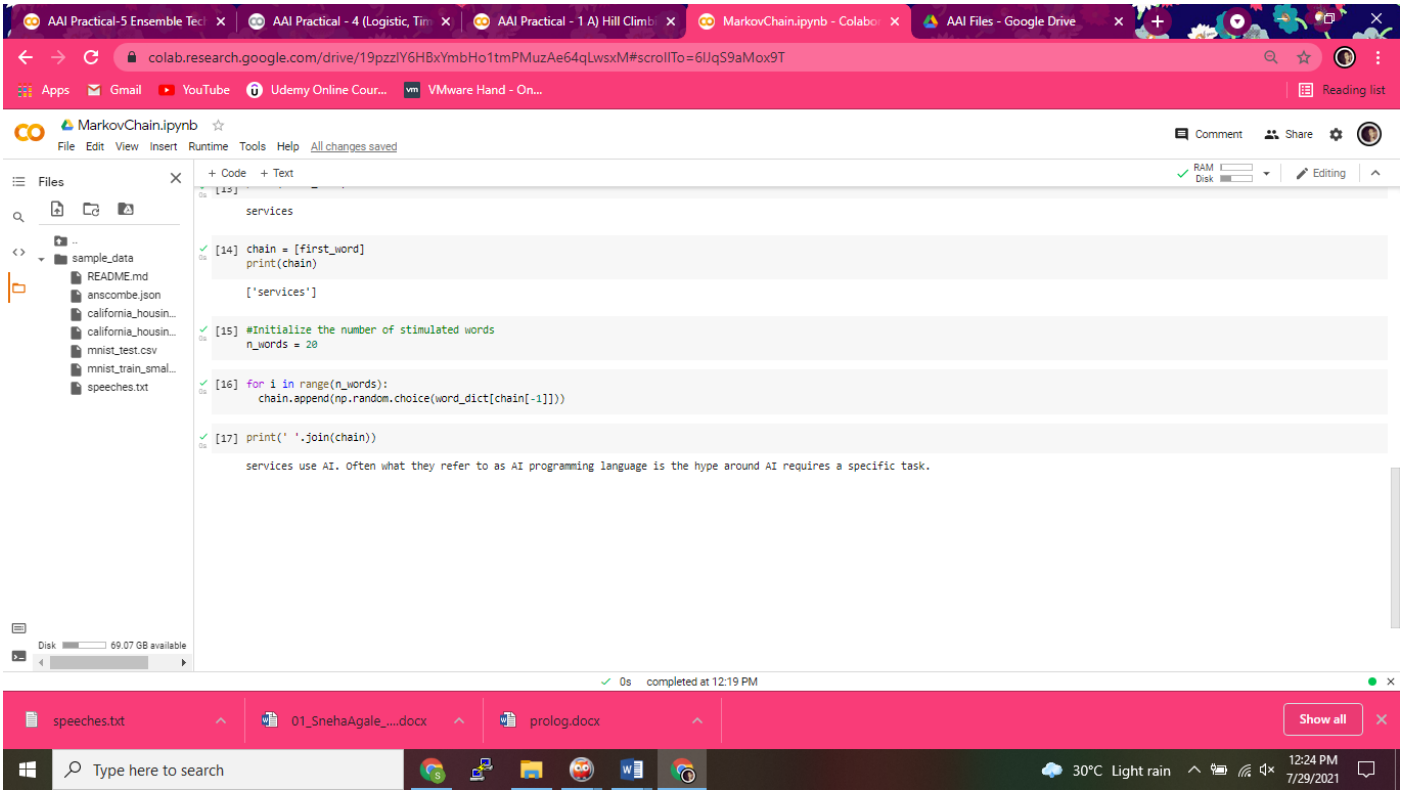
CODE:

```
import numpy as np
speech = open('/content/sample_data/speeches.txt').read()
print(speech)
corpus = speech.split()
print(corpus)
def make_pairs(corpus):
  for i in range(len(corpus) - 1):
    yield (corpus[i], corpus[i + 1])

pairs = make_pairs(corpus)
#for pairs in make_pairs(corpus):
    #print(pairs)
word_dict = {}
for word_1, word_2 in pairs:
  if word_1 in word_dict.keys():
    word_dict[word_1].append(word_2)
  else:
    word_dict[word_1] = [word_2]

print(word_dict)
#randomly pick the first word
first_word = np.random.choice(corpus)
print(first_word)
chain = [first_word]
print(chain)
#Initialize the number of stimulated words
n_words = 20
for i in range(n_words):
  chain.append(np.random.choice(word_dict[chain[-1]]))
print(' '.join(chain))
```

OUTPUT:


AAI Practical-5 Ensemble Tec
AAI Practical - 4 (Logistic, Tim
AAI Practical - 1 A) Hill Climb
MarkovChain.ipynb - Colabo
AAI Files - Google Drive
colab.research.google.com/drive/19pzzlY6HBxYmbHo1tmPMuzAe64qLwsxM#scrollTo=6IJqS9aMox9T
Apps
Gmail
YouTube
Udemy Online Cour...
VMware Hand - On...
Reading list
MarkovChain.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
README.md
anscombe.json
california_housin...
california_housin...
mnist_test.csv
mnist_train_smal...
speeches.txt
Disk 69.07 GB available
+ Code + Text
RAM Disk
Editing
Markov Chain Text Generator
Problem Statement: To apply Markov Property and create a Markov Model that can generate text simulations
Logic: Apply Markov Property to generate speech by considering each word used in the speech and for each word, create a dictionary of words that are used next.
[8] import numpy as np
[9] speech = open('/content/sample_data/speeches.txt').read()
print(speech)
What is artificial intelligence?
Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processi
How does AI work?
As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as mach
In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future stat
AI programming focuses on three cognitive skills: learning, reasoning and self-correction.
Learning processes. This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, prov
[10] corpus = speech.split()
print(corpus)
['What', 'is', 'artificial', 'intelligence?', 'Artificial', 'intelligence', 'is', 'the', 'simulation', 'of', 'human', 'intelligence', 'processes', 'by', 'machines,', 'especially', 'computer', '
0s completed at 12:19 PM
speeches.txt
01_SnehaAgale_....docx
prolog.docx
Show all
Type here to search
30°C AQI 262
12:24 PM
7/29/2021


AAI Practical-5 Ensemble Tec
AAI Practical - 4 (Logistic, Tim
AAI Practical - 1 A) Hill Climb
MarkovChain.ipynb - Colabo
AAI Files - Google Drive
colab.research.google.com/drive/19pzzlY6HBxYmbHo1tmPMuzAe64qLwsxM#scrollTo=6IJqS9aMox9T
Apps
Gmail
YouTube
Udemy Online Cour...
VMware Hand - On...
Reading list
MarkovChain.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
README.md
anscombe.json
california_housin...
california_housin...
mnist_test.csv
mnist_train_smal...
speeches.txt
Disk 69.07 GB available
+ Code + Text
RAM Disk
Editing
[11] def make_pairs(corpus):
for i in range(len(corpus) - 1):
yield (corpus[i], corpus[i + 1])
pairs = make_pairs(corpus)
#for pairs in make_pairs(corpus):
#print(pairs)
[12] word_dict = {}
for word_1, word_2 in pairs:
if word_1 in word_dict.keys():
word_dict[word_1].append(word_2)
else:
word_dict[word_1] = [word_2]
print(word_dict)
{'What': ['is'], 'is': ['artificial', 'the', 'simply', 'synonymous', 'fed'], 'artificial': ['intelligence?'], 'intelligence?': ['Artificial'], 'Artificial': ['intelligence'], 'intelligence': ['
[13] #randomly pick the first word
first_word = np.random.choice(corpus)
print(first_word)
services
[14] chain = [first_word]
print(chain)
['services']
0s completed at 12:19 PM
speeches.txt
01_SnehaAgale_....docx
prolog.docx
Show all
Type here to search
30°C Light rain
12:24 PM
7/29/2021



Google Colab Notebook Link:

https://colab.research.google.com/drive/19pzzlY6HBxYmbHo1tmPMuzAe64qLwsxM?usp=sharing

Drive link of File:

https://drive.google.com/file/d/1EzH5ePU8UqzFOvoOAie_cTraPYcE4L1f/view?usp=sharing

PRACTICAL – 3

Reinforcement Learning: Monte Carlo Prediction

A Monte Carlo simulation is a useful tool for predicting future results by calculating a formula multiple times with different random inputs. At its simplest level, a Monte Carlo analysis (or simulation) involves running many scenarios with different random inputs and summarizing the distribution of the results. Using the commissions analysis, we can continue the manual process we started above but run the program 100's or even 1000's of times and we will get a distribution of potential commission amounts. This distribution can inform the likelihood that the expense will be within a certain window. At the end of the day, this is a prediction so we will likely never predict it exactly. We can develop a more informed idea about the potential risk of under or over budgeting.

There are two components to running a Monte Carlo simulation:

- the equation to evaluate
- the random variables for the input

This is a process you can execute in Excel but it is not simple to do without some VBA or potentially expensive third party plugins. Using numpy and pandas to build a model and generate multiple potential results and analyze them is relatively straightforward. The other added benefit is that analysts can run many scenarios by changing the inputs and can move on to much more sophisticated models in the future if the needs arise. Finally, the results can be shared with non technical users and facilitate discussions around the uncertainty of the final results.

MCS is a computer-intensive technique and, for this reason, its adoption has gone hand-in-hand with the increasing cost effectiveness of computers. It is very widely used in areas as diverse as traffic flow simulation, deep coal mining logistics, post office queuing protocols, hazard analysis of chemical plants, marketing and CRM, and risk analysis in global finance.

MCS comes in many flavours, but perhaps the simplest — and the one we shall be studying — is the so-called 'discrete event-based simulation', which has the following key ingredients

A clock: which measures time and is used to determine when things happen in the simulation.

Entities: are things about which we need to record information (attributes) during the course of the simulation. For example, customers in a post office would be modelled as 'entities', as would the counter clerks. Entities may interact — for example, when a customer is being served.

Events: are measured by the clock and coincide with changes in the attributes of entities, or the birth and death of entities. For example, a customer's arrival in the post office would be an event.

Processes: a process is a set of rules, which is triggered by an event, which may generate subsequent events and which is associated with one or more entities. For example, the serving of a customer by a counter clerk would be a process.

Random Number Generator: is used in conjunction with various statistical distributions to generate events and execute processes. The Generator underpins the probabilistic (hence 'Monte Carlo') nature of the simulation.

Event Scheduler: is used in conjunction with the other elements of the model to manage the generation of events and the execution of processes. The event scheduler is usually rule based.

The evolution of entity attributes is tracked over time and stored for analysis

MCS is run (usually on a computer) with a set of starting entities and attribute values and its evolution tracked over time by storing the attribute values of selected entities in a data set for subsequent analysis. Some summary statistics are also tracked over time (eg queue size in the case of our post office example). Depending on the assumptions being evaluated, multiple runs may be needed, each with a different random number stream so that the 'volatility' and 'average' behaviour of the underlying process can be understood.

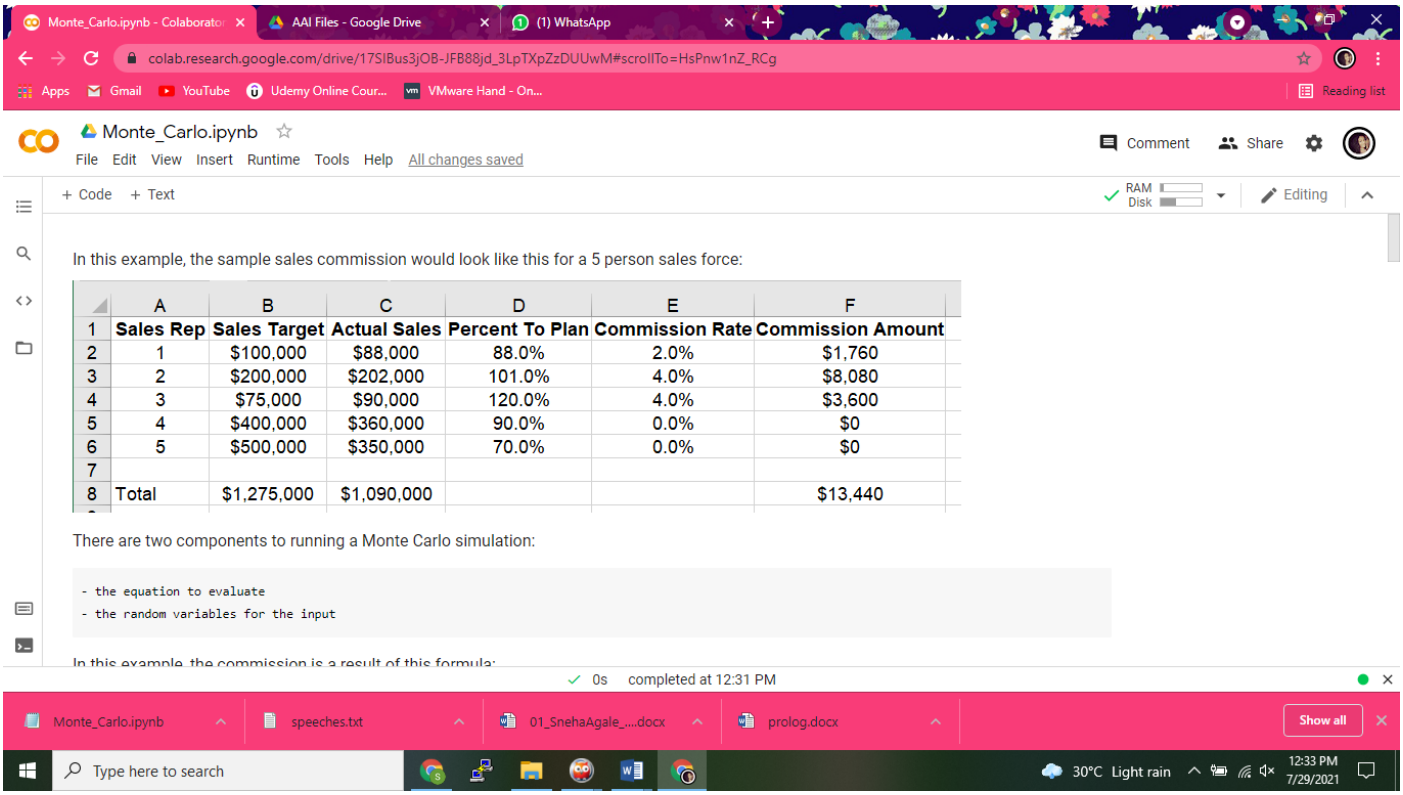
MCS provides a way for analysing complicated systems in which chance plays a key role. In marketing analytics, it has a range of important applications, including the integration of customer behaviour models, the modelling of customer service processes and the evaluation and selection of analytical software tools.

CODE:

```
import pandas as pd
import numpy as np
import seaborn as sns

sns.set_style('whitegrid')


# Define the variables for the Percent to target based on historical results
avg = 1
std_dev = .1
num_reps = 500
#num_simulations = 1000

# Show an example of calculating the percent to target
pct_to_target = np.random.normal(avg, std_dev, num_reps).round(2)
pct_to_target[0:10]



# Another example for the sales target distribution
sales_target_values = [75_000, 100_000, 200_000, 300_000, 400_000, 500_000]
sales_target_prob = [.3, .3, .2, .1, .05, .05]
sales_target = np.random.choice(sales_target_values, num_reps, p=sales_target_prob)
```

```
sales_target[0:10]



# Show how to create the dataframe
df = pd.DataFrame(index=range(num_reps), data={'Pct_To_Target': pct_to_target,
                                               'Sales_Target': sales_target})
df.head()



# Simple histogram to confirm distribution looks as expected
df['Pct_To_Target'].plot(kind='hist', title='Historical % to Target Distribution')



# Look at the sales target distribution
df['Sales_Target'].plot(kind='hist', title='Historical Sales Target Distribution')


# Back into the actual sales amount
df['Sales'] = df['Pct_To_Target'] * df['Sales_Target']


def calc_commission_rate(x):
    """ Return the commission rate based on the table:
    0-90% = 2%
    91-99% = 3%
    >= 100 = 4%
    """
    if x <= .90:
        return .02
    if x <= .99:
        return .03
    else:
        return .04



df['Commission_Rate'] = df['Pct_To_Target'].apply(calc_commission_rate)
df.head()


# Calculate the commissions
df['Commission_Amount'] = df['Commission_Rate'] * df['Sales']
df.head()


print(df['Sales'].sum(), df['Commission_Amount'].sum(), df['Sales_Target'].sum())
```

OUTPUT:



Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

In this example, the sample sales commission would look like this for a 5 person sales force:

	A	B	C	D	E	F
1	**Sales Rep**	**Sales Target**	**Actual Sales**	**Percent To Plan**	**Commission Rate**	**Commission Amount**
2	1	$100,000	$88,000	88.0%	2.0%	$1,760
3	2	$200,000	$202,000	101.0%	4.0%	$8,080
4	3	$75,000	$90,000	120.0%	4.0%	$3,600
5	4	$400,000	$360,000	90.0%	0.0%	$0
6	5	$500,000	$350,000	70.0%	0.0%	$0
7						
8	Total	$1,275,000	$1,090,000			$13,440

There are two components to running a Monte Carlo simulation:

```
- the equation to evaluate
- the random variables for the input
```

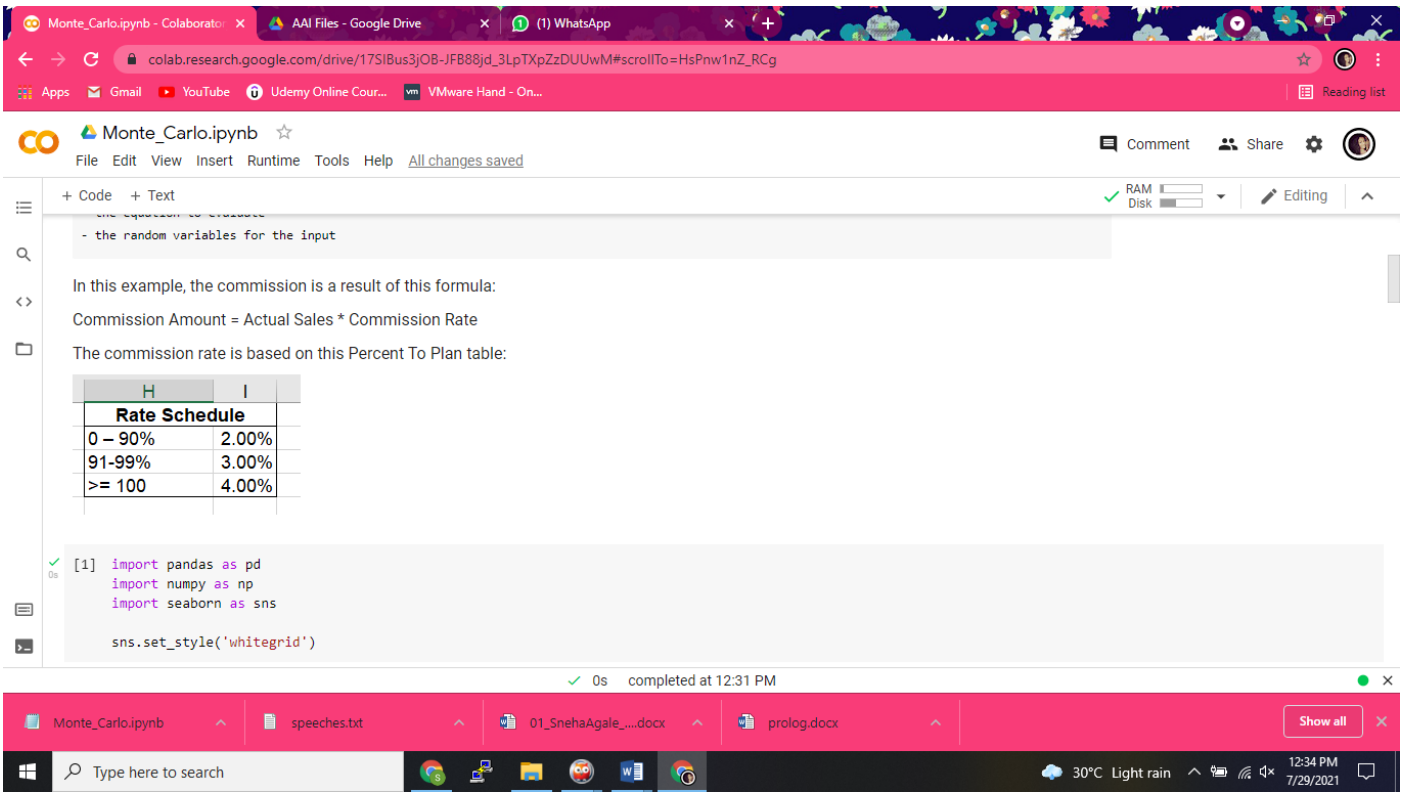
In this example, the commission is a result of this formula:

0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search

30°C Light rain 12:33 PM 7/29/2021



Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
- the random variables for the input
```

In this example, the commission is a result of this formula:

Commission Amount = Actual Sales * Commission Rate

The commission rate is based on this Percent To Plan table:

H	I
Rate Schedule	
0 – 90%	2.00%
91-99%	3.00%
>= 100	4.00%

```
[1] import pandas as pd
    import numpy as np
    import seaborn as sns

    sns.set_style('whitegrid')
```

0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search

30°C Light rain 12:34 PM 7/29/2021

Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Apps | Gmail | YouTube | Udemy Online Cour... | VMware Hand - On... | Reading list

Monte_Carlo.ipynb

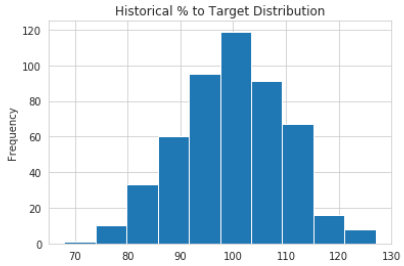
File Edit View Insert Runtime Tools Help All changes saved

Comment | Share

+ Code + Text

RAM Disk | Editing

we can look at a typical historical distribution of percent to target:



This distribution looks like a normal distribution with a mean of 100% and standard deviation of 10%. This insight is useful because we can model our input variable distribution so that it is similar to our real world experience. An approach for predicting next year's commission expense.

0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search

30°C AQI 262 12:34 PM 7/29/2021

Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Apps | Gmail | YouTube | Udemy Online Cour... | VMware Hand - On... | Reading list

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment | Share

+ Code + Text

RAM Disk | Editing

[2]
```
# Define the variables for the Percent to target based on historical results
avg = 1
std_dev = .1
num_reps = 500
#num_simulations = 1000

# Show an example of calculating the percent to target
pct_to_target = np.random.normal(avg, std_dev, num_reps).round(2)
pct_to_target[0:10]
```

```
array([0.98, 1.08, 0.91, 0.94, 0.91, 1.07, 0.91, 0.96, 0.95, 1.  ])
```

[3]
```
# Another example for the sales target distribution
sales_target_values = [75_000, 100_000, 200_000, 300_000, 400_000, 500_000]
sales_target_prob = [.3, .3, .2, .1, .05, .05]
sales_target = np.random.choice(sales_target_values, num_reps, p=sales_target_prob)
sales_target[0:10]
```

```
array([100000, 100000,  75000,  75000, 300000, 400000, 500000,  75000,
       75000, 100000])
```

0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search

30°C AQI 262 12:35 PM 7/29/2021

Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Apps | Gmail | YouTube | Udemy Online Cour... | VMware Hand - On... | Reading list

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment | Share

+ Code + Text | RAM Disk | Editing

```
array([100000, 100000,  75000,  75000, 300000, 400000, 500000,  75000,
        75000, 100000])
```

[4]
```
# Show how to create the dataframe
df = pd.DataFrame(index=range(num_reps), data={'Pct_To_Target': pct_to_target,
                                               'Sales_Target': sales_target})

df.head()
```

	Pct_To_Target	Sales_Target
0	0.98	100000
1	1.08	100000
2	0.91	75000
3	0.94	75000
4	0.91	300000

[5]
```
# Simple histogram to confirm distribution looks as expected
df['Pct_To_Target'].plot(kind='hist', title='Historical % to Target Distribution')
```

0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search | 30°C AQI 262 | 12:35 PM 7/29/2021

Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Apps | Gmail | YouTube | Udemy Online Cour... | VMware Hand - On... | Reading list

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment | Share

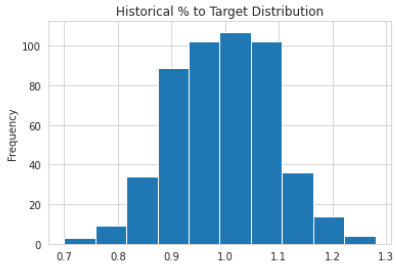
+ Code + Text | RAM Disk | Editing

[4]

4	0.91	300000

[5]
```
# Simple histogram to confirm distribution looks as expected
df['Pct_To_Target'].plot(kind='hist', title='Historical % to Target Distribution')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30a9103550>
```



0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search | 30°C AQI 262 | 12:35 PM 7/29/2021

Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Apps | Gmail | YouTube | Udemy Online Cour... | VMware Hand - On... | Reading list

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment | Share

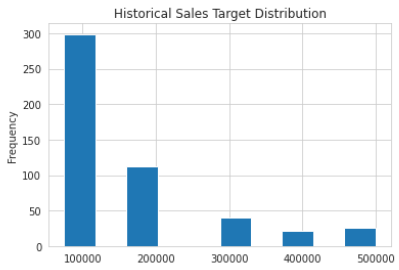
+ Code + Text

RAM Disk | Editing

[5]


```
[6] # Look at the sales target distribution
    df['Sales_Target'].plot(kind='hist', title='Historical Sales Target Distribution')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30a8fdd090>
```



0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search

30°C AQI 262 12:35 PM 7/29/2021

Monte_Carlo.ipynb - Colaborator | AAI Files - Google Drive | (1) WhatsApp

colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM#scrollTo=HsPnw1nZ_RCg

Apps | Gmail | YouTube | Udemy Online Cour... | VMware Hand - On... | Reading list

Monte_Carlo.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment | Share

+ Code + Text

RAM Disk | Editing

[6]


```
[7] # Back into the actual sales amount
    df['Sales'] = df['Pct_To_Target'] * df['Sales_Target']
```

```
[8] def calc_commission_rate(x):
        """ Return the commission rate based on the table:
        0-90% = 2%
        91-99% = 3%
        >= 100 = 4%
        """
        if x <= .90:
            return .02
        if x <= .99:
            return .03
        else:
            return .04
```

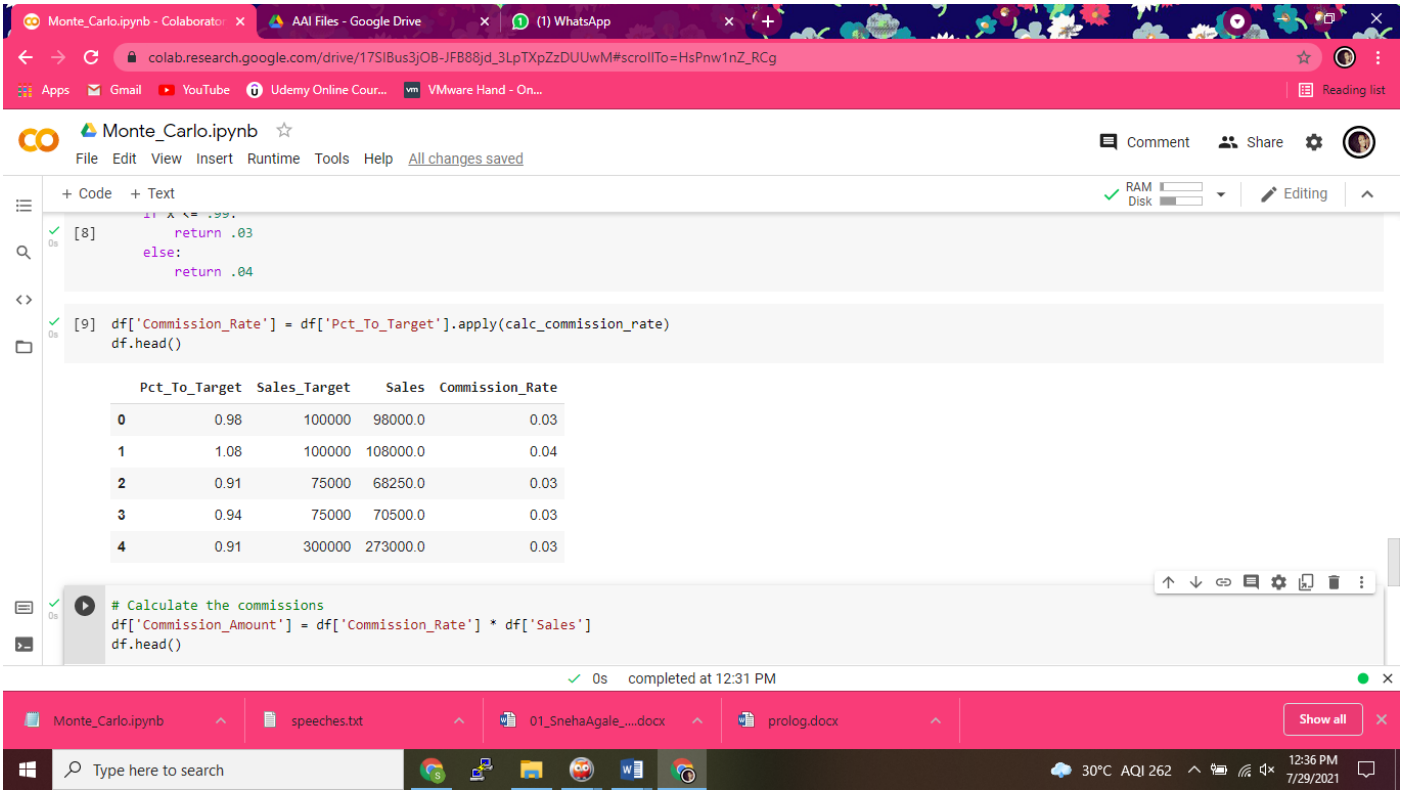
```
[9] df['Commission_Rate'] = df['Pct_To_Target'].apply(calc_commission_rate)
```

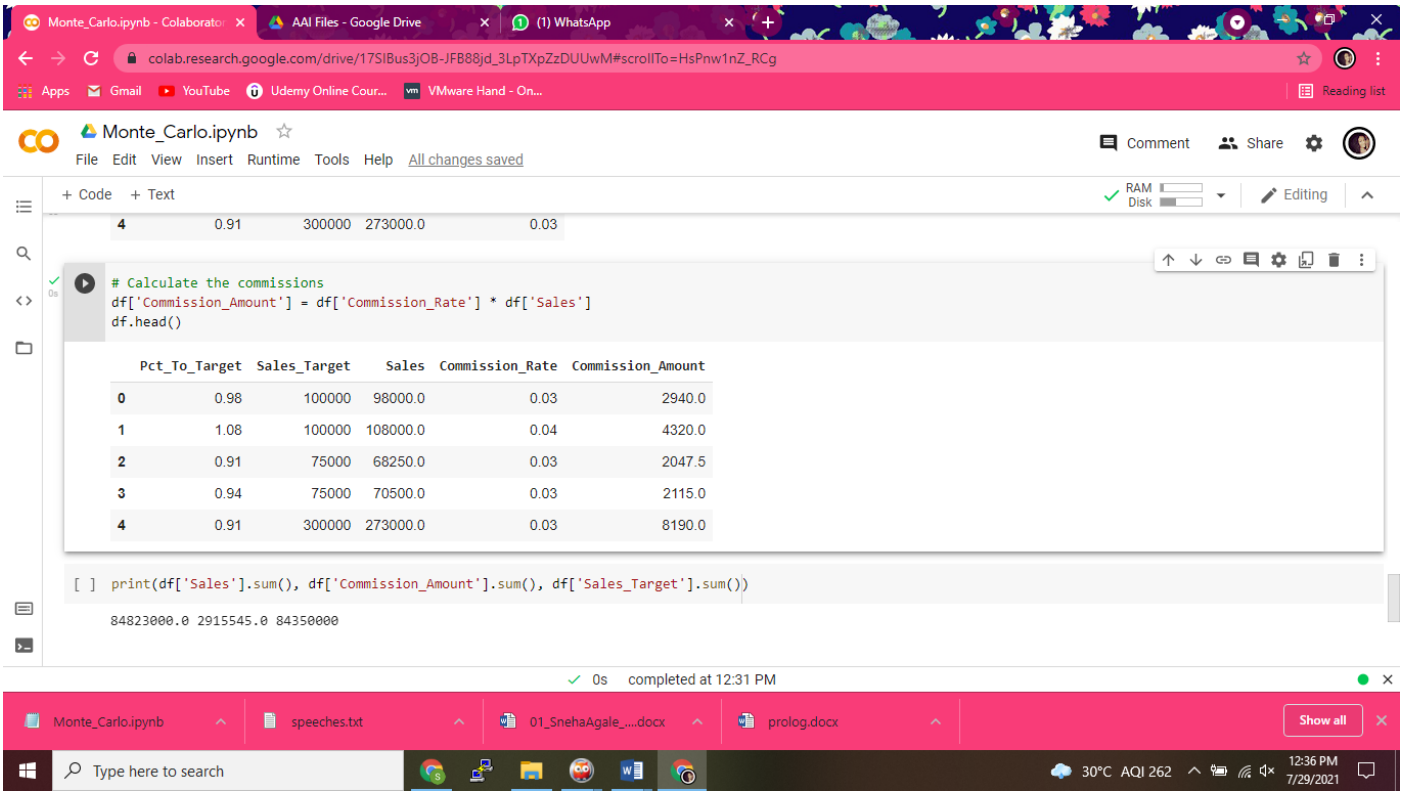
0s completed at 12:31 PM

Monte_Carlo.ipynb | speeches.txt | 01_SnehaAgale_....docx | prolog.docx | Show all

Type here to search

30°C AQI 262 12:35 PM 7/29/2021





Google Colab Notebook Link:

https://colab.research.google.com/drive/17SIBus3jOB-JFB88jd_3LpTXpZzDUUwM?usp=sharing

PRACTICAL – 4

Predictive Analytics - Forecasting (Logistic, Time Series - ARIMA, Case Study)

A time series is a sequence where a metric is recorded over regular time intervals. Depending on the frequency, a time series can be of yearly (ex: annual budget), quarterly (ex: expenses), monthly (ex: air traffic), weekly (ex: sales qty), daily (ex: weather), hourly (ex: stocks price), minutes (ex: inbound calls in a call canter) and even seconds wise (ex: web traffic). Forecasting a time series (like demand and sales) is often of tremendous commercial value.

Forecasting a time series can be broadly divided into two types:

If you use only the previous values of the time series to predict its future values, it is called **Univariate Time Series Forecasting.**

And if you use predictors other than the series (a.k.a exogenous variables) to

forecast it is called **Multi Variate Time Series Forecasting**.

ARIMA, short for 'Auto Regressive Integrated Moving Average' is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models.

An ARIMA model is characterized by 3 terms: p, d, q

where,

p is the order of the AR term

q is the order of the MA term

d is the number of differencing required to make the time series stationary

'Auto Regressive' in ARIMA means it is a **linear regression model** that uses its

own lags as predictors. The most common approach is to difference it. That is, subtract the previous value from the current value. Sometimes, depending on the

complexity of the series, more than one differencing may be needed.

The value of d, therefore, is the minimum number of differencing needed to make

the series stationary. And if the time series is already stationary, then d = 0.

'p' is the order of the 'Auto Regressive' (AR) term. It refers to the number of lags

of Y to be used as predictors. And 'q' is the order of the 'Moving Average' (MA)

term. It refers to the number of lagged forecast errors that should go into the

ARIMA Model.

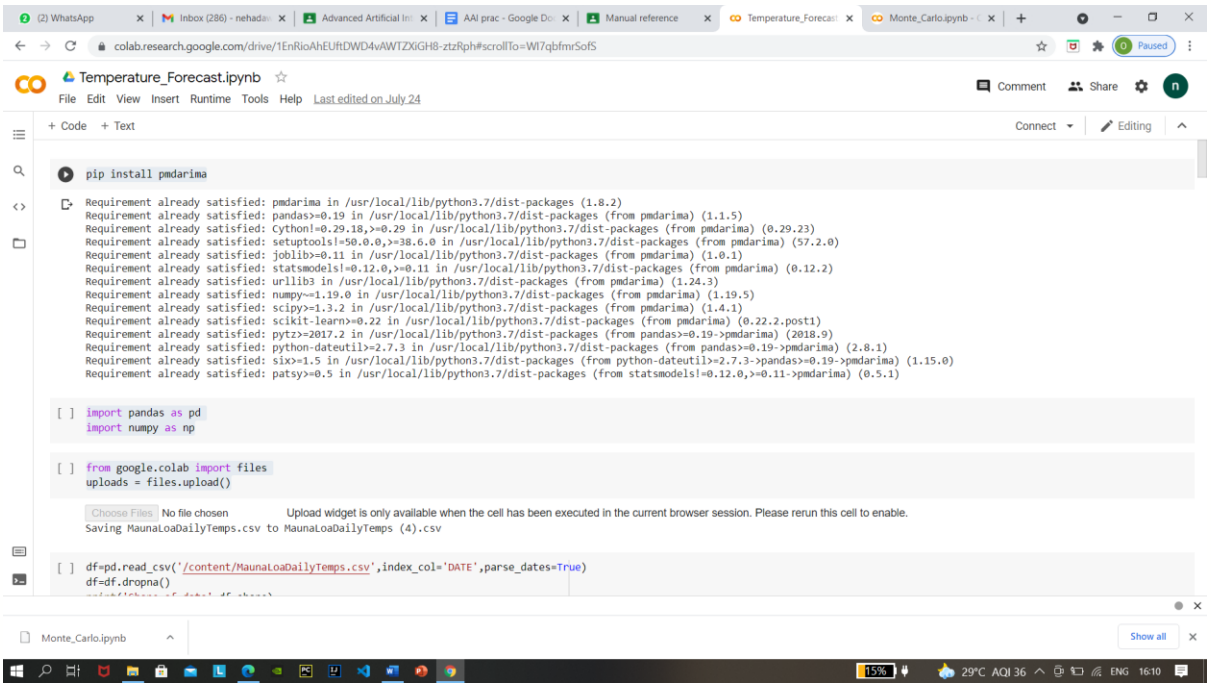
CODE:

```
pip install pmdarima
```

```
import pandas as pd
import numpy as np


from google.colab import files
uploads = files.upload()


df=pd.read_csv('/content/MaunaLoaDailyTemps.csv',index_col='DATE',parse_dates=True)
df=df.dropna()
print('Shape of data',df.shape)
df.head()


df['AvgTemp'].plot(figsize=(12,5))


from statsmodels.tsa.stattools import adfuller
def ad_test(dataset):
    dftest = adfuller(dataset, autolag = 'AIC')
    print("1. ADF : ",dftest[0])
    print("2. P-Value : ", dftest[1])
    print("3. Num Of Lags : ", dftest[2])
    print("4. Num Of Observations Used For ADF Regression:",      dftest[3])
    print("5. Critical Values :")
    for key, val in dftest[4].items():
        print("\t",key, ": ", val)
ad_test(df['AvgTemp'])
from pmdarima import auto_arima
stepwise_fit = auto_arima(df['AvgTemp'], trace=True,
suppress_warnings=True)

print(df.shape)
train=df.iloc[:-30]
test=df.iloc[-30:]
print(train.shape,test.shape)

from statsmodels.tsa.arima_model import ARIMA
model=ARIMA(train['AvgTemp'],order=(1,0,5))
model=model.fit()
model.summary()

start=len(train)
end=len(train)+len(test)-1
pred=model.predict(start=start,end=end,typ='levels')
#print(pred)
pred.index=df.index[start:end+1]
print(pred)

pred.plot(legend=True)
test['AvgTemp'].plot(legend=True)

test['AvgTemp'].mean()
from sklearn.metrics import mean_squared_error
from math import sqrt
```

```
test['AvgTemp'].mean()
rmse=sqrt(mean_squared_error(pred,test['AvgTemp']))
print(rmse)

model2=ARIMA(df['AvgTemp'],order=(1,0,5))
model2=model2.fit()
df.tail()

index_future_dates=pd.date_range(start='2018-12-30' ,end='2019-01-29')
#print(index_future_dates)
pred=model2.predict(start=len(df),end=len(df)+30,typ='levels').rename('ARIMA Predications')
#print(comp_pred)
pred.index=index_future_dates
print(pred)

pred.plot(figsize=(12,5),legend=True)
```

OUTPUT:



Temperature_Forecast.ipynb

File Edit View Insert Runtime Tools Help Last edited on July 24

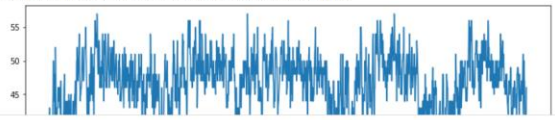
```
df=pd.read_csv('/content/MaunaLoaDailyTemps.csv',index_col='DATE',parse_dates=True)
df=df.dropna()
print('Shape of data',df.shape)
df.head()
```

```
Shape of data (1821, 5)
```

DATE	MinTemp	MaxTemp	AvgTemp	Sunrise	Sunset
2014-01-01	33.0	46.0	40.0	657	1756
2014-01-02	35.0	50.0	43.0	657	1756
2014-01-03	36.0	45.0	41.0	657	1757
2014-01-04	32.0	41.0	37.0	658	1757
2014-01-05	24.0	38.0	31.0	658	1758

```
df['AvgTemp'].plot(figsize=(12,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff056d31c10>
```



Monte_Carlo.ipynb

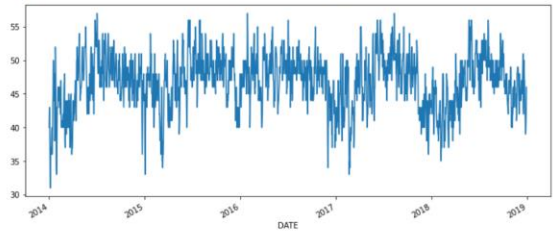
Show all

Temperature_Forecast.ipynb

File Edit View Insert Runtime Tools Help Last edited on July 24

df['AvgTemp'].plot(figsize=(12,5))

<matplotlib.axes._subplots.AxesSubplot at 0x7ff056d31c10>




```
from statsmodels.tsa.stattools import adfuller
def ad_test(dataset):
    dftest = adfuller(dataset, autolag = 'AIC')
    print("1. ADF : ",dftest[0])
    print("2. P-Value : ", dftest[1])
    print("3. Num Of Lags : ", dftest[2])
    print("4. Num Of Observations Used For ADF Regression:",      dftest[3])
    print("5. Critical Values :")
    for key, val in dftest[4].items():
        print("\t",key, ": ", val)
```

Monte_Carlo.ipynb

Show all

Temperature_Forecast.ipynb

File Edit View Insert Runtime Tools Help Last edited on July 24

```
    print("5. Critical Values :")
    for key, val in dftest[4].items():
        print("\t",key, ": ", val)
ad_test(df['AvgTemp'])
```

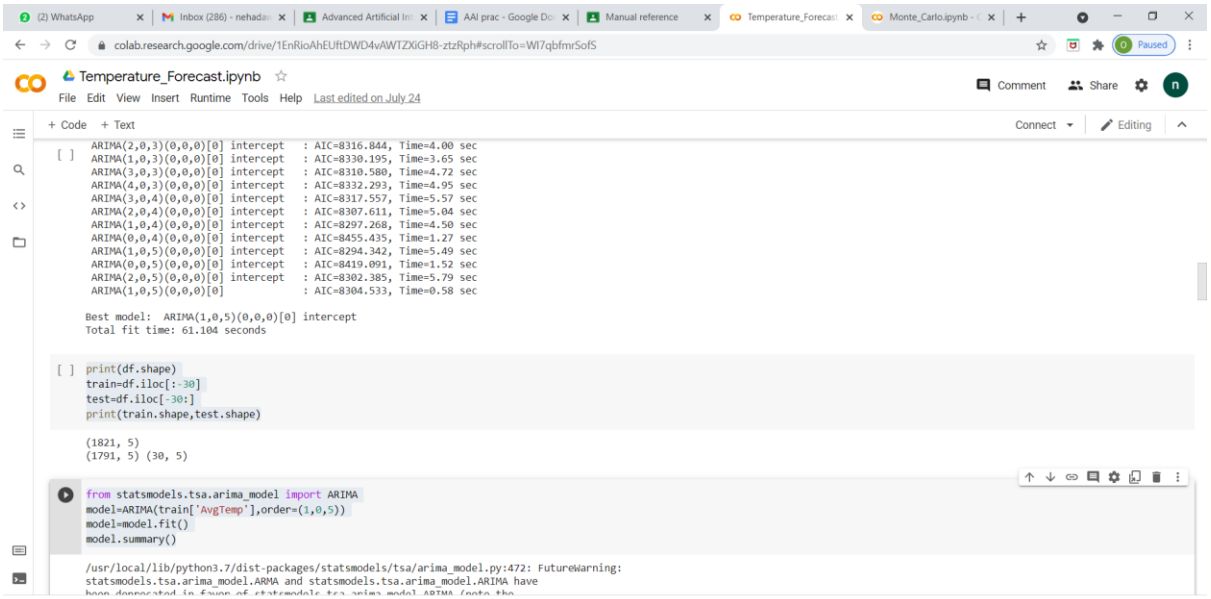
```
1. ADF :  -6.554680125068784
2. P-Value :  8.675937480199322e-09
3. Num Of Lags :  12
4. Num Of Observations Used For ADF Regression: 1808
5. Critical Values :
        1% :  -3.433972018026501
        5% :  -2.8631399192826676
        10% :  -2.5676217442756872
```

```
from pmdarima import auto_arima
stepwise_fit = auto_arima(df['AvgTemp'], trace=True,
suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=8344.973, Time=3.46 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=10347.755, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=8365.701, Time=0.22 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=9136.225, Time=0.40 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=19192.139, Time=0.04 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=8355.947, Time=2.84 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=8356.308, Time=2.96 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=8347.342, Time=3.98 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=8316.844, Time=4.00 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=8330.195, Time=3.65 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=8310.580, Time=4.72 sec
 ARIMA(4,0,3)(0,0,0)[0] intercept   : AIC=8332.293, Time=4.95 sec
 ARIMA(3,0,4)(0,0,0)[0] intercept   : AIC=8317.557, Time=5.57 sec
 ARIMA(2,0,4)(0,0,0)[0] intercept   : AIC=8307.611, Time=5.04 sec
```

Monte_Carlo.ipynb

Show all



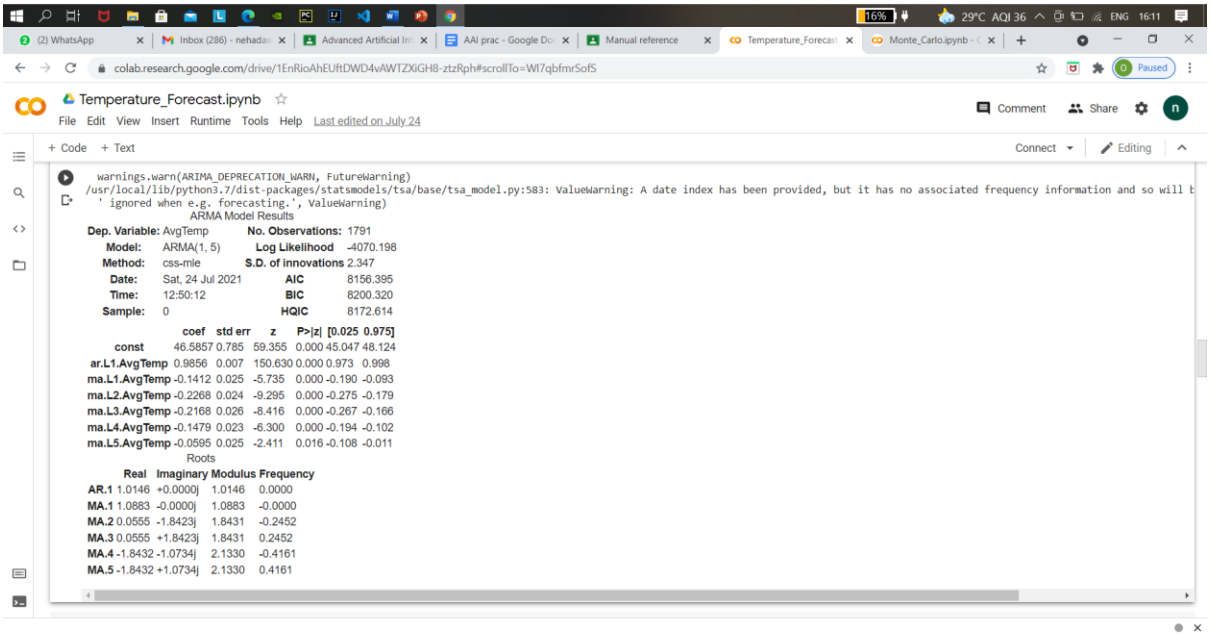
```
ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=8316.844, Time=4.00 sec
ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=8330.195, Time=3.65 sec
ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=8310.580, Time=4.72 sec
ARIMA(4,0,3)(0,0,0)[0] intercept   : AIC=8332.293, Time=4.95 sec
ARIMA(3,0,4)(0,0,0)[0] intercept   : AIC=8317.557, Time=5.57 sec
ARIMA(2,0,4)(0,0,0)[0] intercept   : AIC=8307.611, Time=5.04 sec
ARIMA(1,0,4)(0,0,0)[0] intercept   : AIC=8297.268, Time=4.50 sec
ARIMA(0,0,4)(0,0,0)[0] intercept   : AIC=8455.435, Time=1.27 sec
ARIMA(1,0,5)(0,0,0)[0] intercept   : AIC=8294.342, Time=5.49 sec
ARIMA(0,0,5)(0,0,0)[0] intercept   : AIC=8419.091, Time=1.52 sec
ARIMA(2,0,5)(0,0,0)[0] intercept   : AIC=8302.385, Time=5.79 sec
ARIMA(1,0,5)(0,0,0)[0]             : AIC=8304.533, Time=0.58 sec

Best model:  ARIMA(1,0,5)(0,0,0)[0] intercept
Total fit time: 61.104 seconds
```

```
print(df.shape)
train=df.iloc[:-30]
test=df.iloc[-30:]
print(train.shape,test.shape)
```

```
(1821, 5)
(1791, 5) (30, 5)
```

```
from statsmodels.tsa.arima_model import ARIMA
model=ARIMA(train['AvgTemp'],order=(1,0,5))
model=model.fit()
model.summary()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
```



```
  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:583: ValueWarning: A date index has been provided, but it has no associated frequency information and so will b
  ' ignored when e.g. forecasting.', ValueWarning)
```

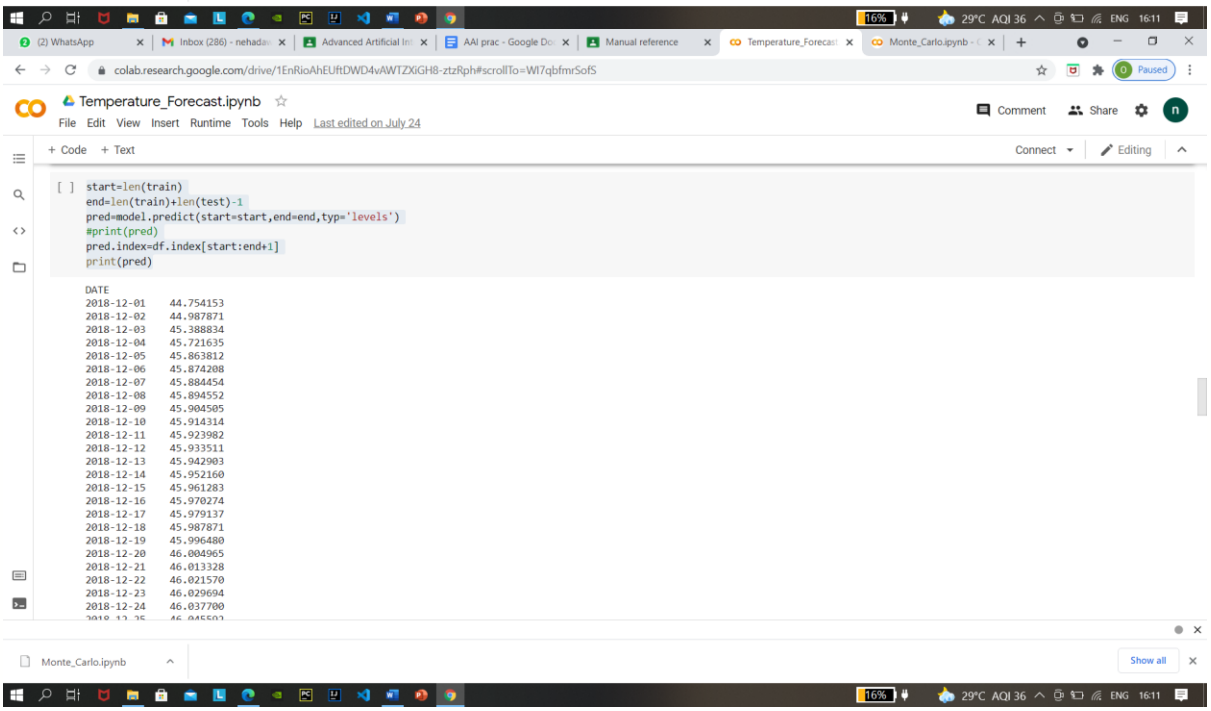
ARMA Model Results

Dep. Variable:	AvgTemp	**No. Observations:**	1791
Model:	ARMA(1, 5)	**Log Likelihood**	-4070.198
Method:	css-mle	**S.D. of innovations**	2.347
Date:	Sat, 24 Jul 2021	**AIC**	8156.395
Time:	12:50:12	**BIC**	8200.320
Sample:	0	**HQIC**	8172.614

	coef	std err	z	P>|z|	[0.025	0.975]
const	46.5857	0.785	59.355	0.000	45.047	48.124
ar.L1.AvgTemp	0.9856	0.007	150.630	0.000	0.973	0.998
ma.L1.AvgTemp	-0.1412	0.025	-5.735	0.000	-0.190	-0.093
ma.L2.AvgTemp	-0.2268	0.024	-9.295	0.000	-0.275	-0.179
ma.L3.AvgTemp	-0.2168	0.026	-8.416	0.000	-0.267	-0.166
ma.L4.AvgTemp	-0.1479	0.023	-6.300	0.000	-0.194	-0.102
ma.L5.AvgTemp	-0.0595	0.025	-2.411	0.016	-0.108	-0.011

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0146	+0.0000j	1.0146	0.0000
MA.1	1.0883	-0.0000j	1.0883	-0.0000
MA.2	0.0555	-1.8423j	1.8431	-0.2452
MA.3	0.0555	+1.8423j	1.8431	0.2452
MA.4	-1.8432	-1.0734j	2.1330	-0.4161
MA.5	-1.8432	+1.0734j	2.1330	0.4161



```
start=len(train)
end=len(train)+len(test)-1
pred=model.predict(start=start,end=end,typ='levels')
#print(pred)
pred.index=df.index[start:end+1]
print(pred)
```

```
DATE
2018-12-01    44.754153
2018-12-02    44.987871
2018-12-03    45.388834
2018-12-04    45.721635
2018-12-05    45.863812
2018-12-06    45.874208
2018-12-07    45.884454
2018-12-08    45.894552
2018-12-09    45.904505
2018-12-10    45.914314
2018-12-11    45.923982
2018-12-12    45.933511
2018-12-13    45.942903
2018-12-14    45.952160
2018-12-15    45.961283
2018-12-16    45.970274
2018-12-17    45.979137
2018-12-18    45.987871
2018-12-19    45.996480
2018-12-20    46.004965
2018-12-21    46.013328
2018-12-22    46.021570
2018-12-23    46.029694
2018-12-24    46.037700
```

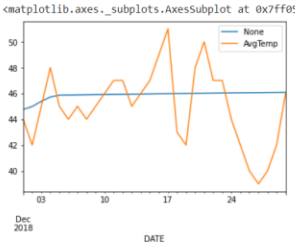
Temperature_Forecast.ipynb

File Edit View Insert Runtime Tools Help Last edited on July 24

```
2018-12-25    46.045592
2018-12-26    46.053369
2018-12-27    46.061035
2018-12-28    46.068590
2018-12-29    46.076037
2018-12-30    46.083376
dtype: float64
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:379: ValueWarning: No supported index is available. Prediction results will be given with an integer index begi
  ValueWarning)
```

```
pred.plot(legend=True)
test['AvgTemp'].plot(legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff056b25190>
```



Monte_Carlo.ipynb

Show all

Temperature_Forecast.ipynb

File Edit View Insert Runtime Tools Help Last edited on July 24

```
test['AvgTemp'].mean()
```

```
45.0
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt
test['AvgTemp'].mean()
rmse=sqrt(mean_squared_error(pred,test['AvgTemp']))
print(rmse)
```

```
3.0004947362496863
```

```
model2=ARIMA(df['AvgTemp'],order=(1,0,5))
model2=model2.fit()
df.tail()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
```

Monte_Carlo.ipynb

Show all

Temperature_Forecast.ipynb

File Edit View Insert Runtime Tools Help Last edited on July 24

```
index_future_dates=pd.date_range(start='2018-12-30' ,end='2019-01-29')
#print(index_future_dates)
pred=model2.predict(start=len(df),end=len(df)+30,typ='levels').rename('ARIMA Predications')
#print(comp_pred)
pred.index=index_future_dates
print(pred)
```

```
2018-12-30    46.418065
2018-12-31    46.113783
2019-01-01    45.617773
2019-01-02    45.249557
2019-01-03    45.116985
2019-01-04    45.136772
2019-01-05    45.156281
2019-01-06    45.175517
2019-01-07    45.194484
2019-01-08    45.213185
2019-01-09    45.231623
2019-01-10    45.249804
2019-01-11    45.267730
2019-01-12    45.285404
2019-01-13    45.302831
2019-01-14    45.320014
2019-01-15    45.336957
2019-01-16    45.353661
2019-01-17    45.370132
2019-01-18    45.386372
2019-01-19    45.402385
2019-01-20    45.418173
2019-01-21    45.433740
2019-01-22    45.449089
```

Monte_Carlo.ipynb

Show all

(2) WhatsApp | Inbox (286) - nehada | Advanced Artificial In | AAI prac - Google Do | Manual reference | Temperature_Forecast | Monte_Carlo.ipynb - C

colab.research.google.com/drive/1EnRioAhEUftDWD4vAWTZXiGH8-ztzRph#scrollTo=Wl7qbfmrSofS

Paused

Temperature_Forecast.ipynb

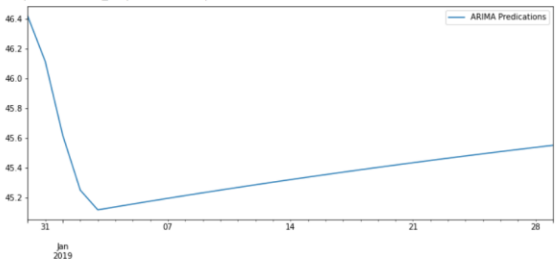
File Edit View Insert Runtime Tools Help Last edited on July 24

Comment Share n

+ Code + Text Connect Editing

```
[ ] pred.plot(figsize=(12,5),legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff056b9bed0>
```



Monte_Carlo.ipynb

Show all

17% 29°C AQI 36 ENG 16:12

PRACTICAL – 5

Ensemble Techniques (Boosting, Bagging).

The two most popular methods for combining the predictions from different models are:

- Bagging. Building multiple models (typically of the same type) from different subsamples of the training dataset.
- Boosting. Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.

Bagging Algorithms

Bootstrap Aggregation or bagging involves taking multiple samples from your training dataset (with replacement) and training a model for each sample.The final output prediction is averaged across the predictions of all of the sub-models.

The three bagging models covered in this section are as follows:

1. Bagged Decision Trees
2. Random Forest
3. Extra Trees

1. Bagged Decision Trees

Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning.

In the example below see an example of using the BaggingClassifier with the Classification and Regression Trees algorithm (DecisionTreeClassifier). A total of 100 trees are created.

CODE:

```
import pandas as pd
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier


url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
df = pd.read_csv(url, names=names)
df.head()  #show the first five elements of the dataset
df.tail()  #show the last five elements of the dataset
array=df.values
array
#splitting the dataset in x i.e. features and y i.e. target variables
x = array[:,0:8]
```

```
y = array[:,8]
x
y
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model=DecisionTreeClassifier()
n_trees=100
model1 = BaggingClassifier(base_estimator=model, n_estimators=n_trees, random_state=seed)
results = model_selection.cross_val_score(model1, x, y, cv=kfold)
print(results.mean())
```

OUTPUT:

```
0.770745044429255
```

2. Random Forest

Random forest is an extension of bagged decision trees.

Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split.

You can construct a Random Forest model for classification using the RandomForestClassifier class.

The example below provides an example of Random Forest for classification with 100 trees and split points chosen from a random selection of 3 features.

CODE:

```
import pandas
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier


url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age' 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
a = array[:,0:8]
b = array[:,7]
seed = 6
num_trees = 100  #number of trees
max_features = 3  #maximum features
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, a, b, cv=kfold)
print(results.mean())
```

OUTPUT:

```
1.0
```

3. Extra Trees

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset.

You can construct an Extra Trees model for classification using

the ExtraTreesClassifier class.
The example below provides a demonstration of extra trees with the number of trees set to 100 and splits chosen from 7 random features.

CODE:

```
import pandas as pd
from sklearn import model_selection
from sklearn.ensemble import ExtraTreesClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
df= pd.read_csv(url, names=names)
array=df.values
x=array[:,0:8]
y=array[:,8]
seed=7
num_trees = 100
max_features = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = ExtraTreesClassifier(n_estimators=num_trees,
max_features=max_features)
results = model_selection.cross_val_score(model, x, y, cv=kfold)
print(results.mean())
```

OUTPUT:

```
0.772095010252905
```

Boosting Algorithms

Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence.

Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction.

The two most common boosting ensemble machine learning algorithms are:

1. **AdaBoost**
2. **Stochastic Gradient Boosting**

1. AdaBoost

AdaBoost was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay or or less attention to them in the construction of subsequent models.

You can construct an AdaBoost model for classification using the AdaBoostClassifier class.

The example below demonstrates the construction of 30 decision trees in sequence using the AdaBoost algorithm.

CODE:

```
import pandas as pd
```

```
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

OUTPUT:

```
0.760457963089542
```

2. Stochastic Gradient Boosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps of the the best techniques available for improving performance via ensembles.

You can construct a Gradient Boosting model for classification using the GradientBoostingClassifier class.

The example below demonstrates Stochastic Gradient Boosting for classification with 100 trees.

CODE:

```
import pandas as pd
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
df = pd.read_csv(url, names=names)
array = df.values
x = array[:,0:8]
y = array[:,8]
seed = 7
ntrees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=ntrees, random_state=seed)
results = model_selection.cross_val_score(model, x, y, cv=kfold)
print(results.mean())
```

OUTPUT:

```
0.7681989063568012
```

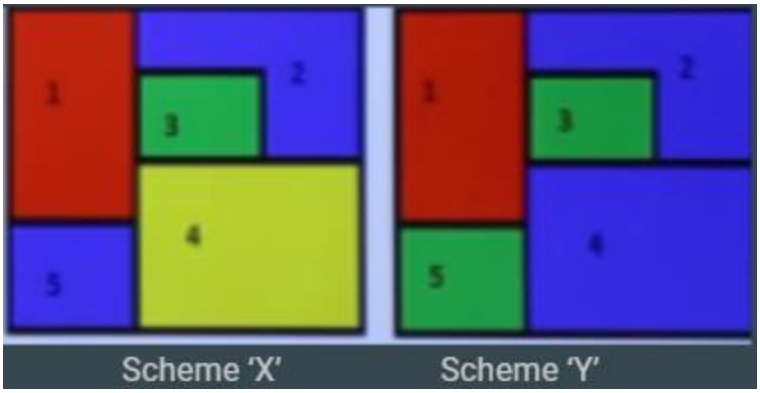
Google Colab Notebook Link:

https://colab.research.google.com/drive/18j9KpqqFPPbDfjgoYq0u-rKnZW6shWmC?usp=sharing

PRACTICAL - 6

Prolog - Color mapping, Cryptarthmatic problem

In mathematics, the famous problem was coloring adjacent planar regions. Two adjacent regions cannot have the same color no matter whatever color we choose. Two regions which share some boundary line are considered adjacent to each other.



Code:(In SWIProlog):
ColorMap.pl

```
%%%%Describing Adjacencies.
adjacent(1,2).      adjacent(2,1).
adjacent(1,3).      adjacent(3,1).
adjacent(1,4).      adjacent(4,1).
adjacent(1,5).      adjacent(5,1).
adjacent(2,3).      adjacent(3,2).
adjacent(2,4).      adjacent(4,2).
adjacent(3,4).      adjacent(4,3).
adjacent(4,5).      adjacent(5,4).

%%%%Assigning Color.
color(1, orange, x).    color(1, orange, y).
color(2, blue, x).  color(2, blue, y).
color(3, green, x).    color(3, green, y).
color(4, yellow, x).   color(4, blue, y).
color(5, blue, x).  color(5, green, y).

%%%Checking for conflict in color.
conflict(R1, R2, Coloring) :-
adjacent(R1, R2),
color(R1, Color, Coloring),
color(R2, Color, Coloring).
```

ColorMap.pl

File Edit Browse Compile Prolog Pce Help

ColorMap.pl

```
%%%%Describing Adjacencies.
adjacent(1,2).          adjacent(2,1).
adjacent(1,3).          adjacent(3,1).
adjacent(1,4).          adjacent(4,1).
adjacent(1,5).          adjacent(5,1).
adjacent(2,3).          adjacent(3,2).
adjacent(2,4).          adjacent(4,2).
adjacent(3,4).          adjacent(4,3).
adjacent(4,5).          adjacent(5,4).

%%%%Assigning Color.
color(1, orange, x).     color(1, orange, y).
color(2, blue, x).   color(2, blue, y).
color(3, green, x).     color(3, green, y).
color(4, yellow, x).    color(4, blue, y).
color(5, blue, x).   color(5, green, y).

%%%Checking for conflict in color.
conflict(R1, R2, Coloring) :-
adjacent(R1, R2),
color(R1, Color, Coloring),
color(R2, Color, Coloring).
```

Line: 23

Type here to search 29°C Rain showers 10:33 AM 7/29/2021

OUTPUT:

SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
|    adjacent(2,3)
|    adjacent(2,3).
ERROR: Syntax error: Operator expected
ERROR: adjacent(2,3)
ERROR: ** here **
ERROR:
adjacent(2,3) ,
?- adjacent(2,3).
true.

?- adjacent(2,5).
false.

?- adjacent(5,2).
false.

?- adjacent(5,4).
true.

?- conflict(R1,R2,x).
false.

?- conflict(R1,R2,y),color(R1,Z,y).
R1 = 2,
R2 = 4,
Z = blue
```

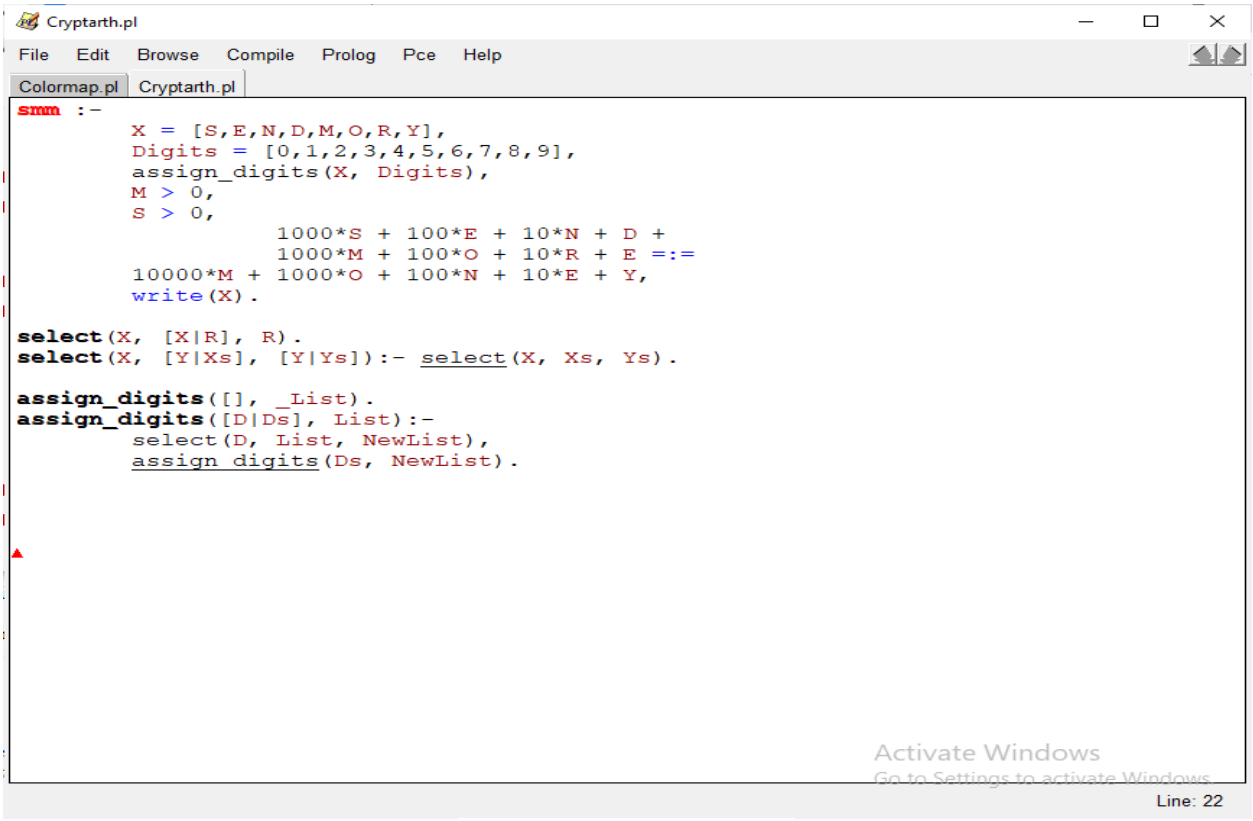
Type here to search 29°C Rain showers 10:33 AM 7/29/2021

Cryptarthmic Problem:

Cryptarth.pl

CODE:

```
smm :-
        X = [S,E,N,D,M,O,R,Y],
        Digits = [0,1,2,3,4,5,6,7,8,9],
        assign_digits(X, Digits),
        M > 0,
        S > 0,
                1000*S + 100*E + 10*N + D +
                1000*M + 100*O + 10*R + E =:=
       10000*M + 1000*O + 100*N + 10*E + Y,
       write(X).
select(X, [X|R],  R).
select(X, [Y|Xs], [Y|Ys]):- select(X, Xs, Ys).
assign_digits([], _List).
assign_digits([D|Ds], List):-
    select(D, List, NewList),
    assign_digits(Ds, NewList).
```

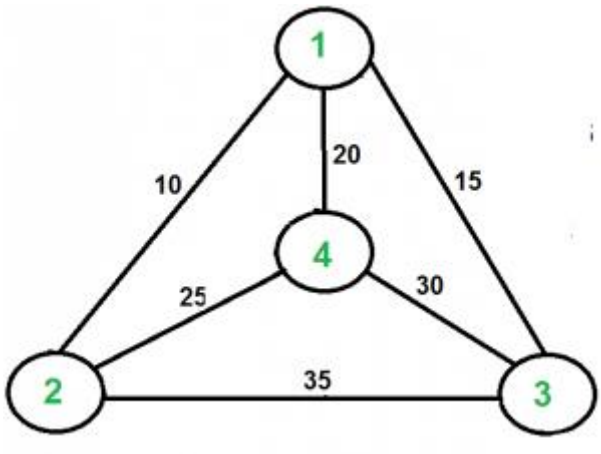



```
?-
|    smm.
[9,5,6,7,1,0,8,2]
true
```

PRACTICAL - 7

Travelling Salesman Problem

Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.
For example, consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.
The problem is a famous NP-hard problem. There is no polynomial-time known solution for this problem.



Naive Solution:

1) Consider city 1 as the starting and ending point.

2) Generate all (n-1)! Permutations of cities.

3) Calculate cost of every permutation and keep track of minimum cost permutation.

4) Return the permutation with minimum cost.

Time Complexity: Θ(n!)

Dynamic Programming:

Let the given set of vertices be {1, 2, 3, 4,....n}. Let us consider 1 as starting and ending point of output. For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point, i as the ending point and all vertices appearing exactly once. Let the cost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance

from i to 1. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values. This looks simple so far. Now the question is how to get cost(i)?

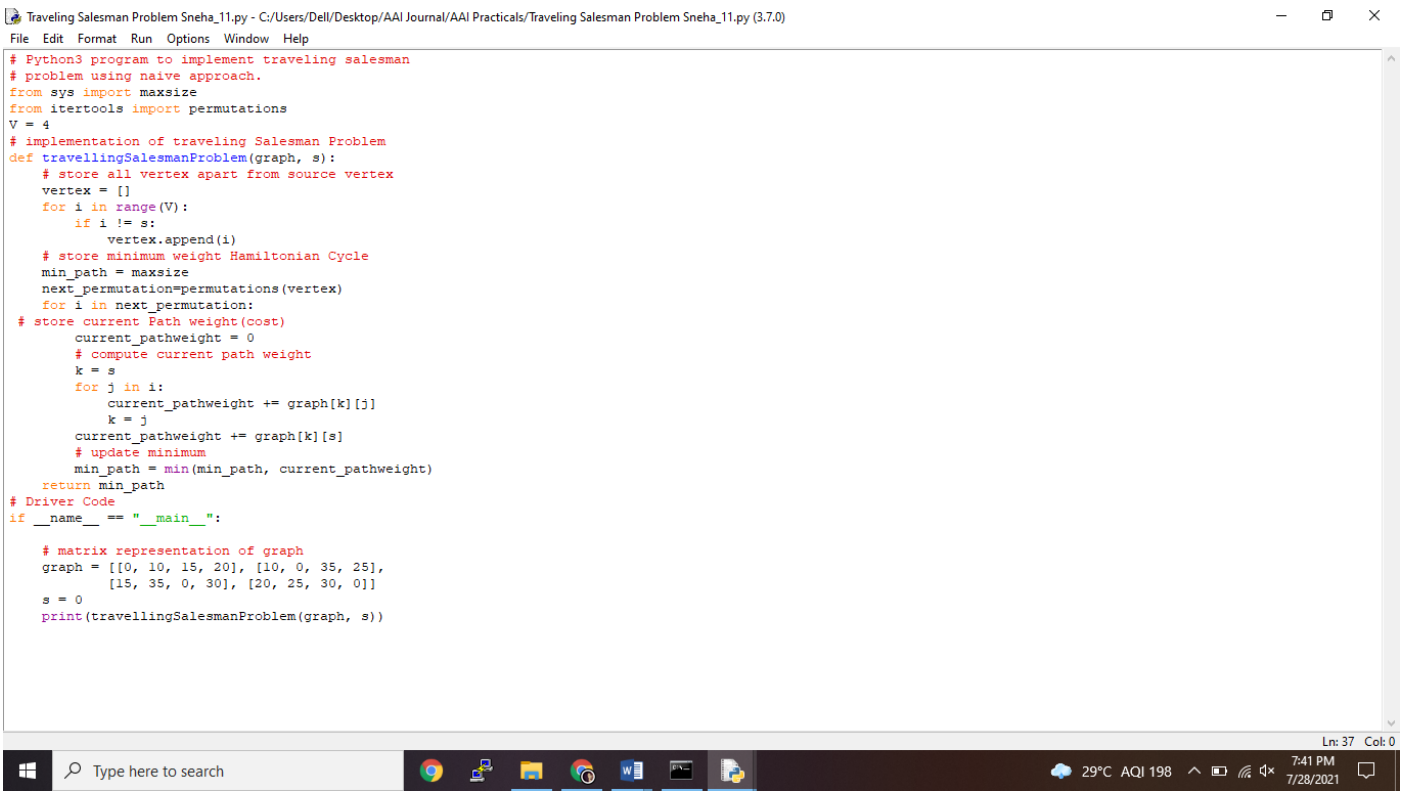
To calculate cost(i) using Dynamic Programming, we need to have some recursive relation in terms of sub-problems. Let us define a term C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i.

We start with all subsets of size 2 and calculate C(S, i) for all subsets where S is the subset, then we calculate C(S, i) for all subsets S of size 3 and so on. Note that 1 must be present in every subset.

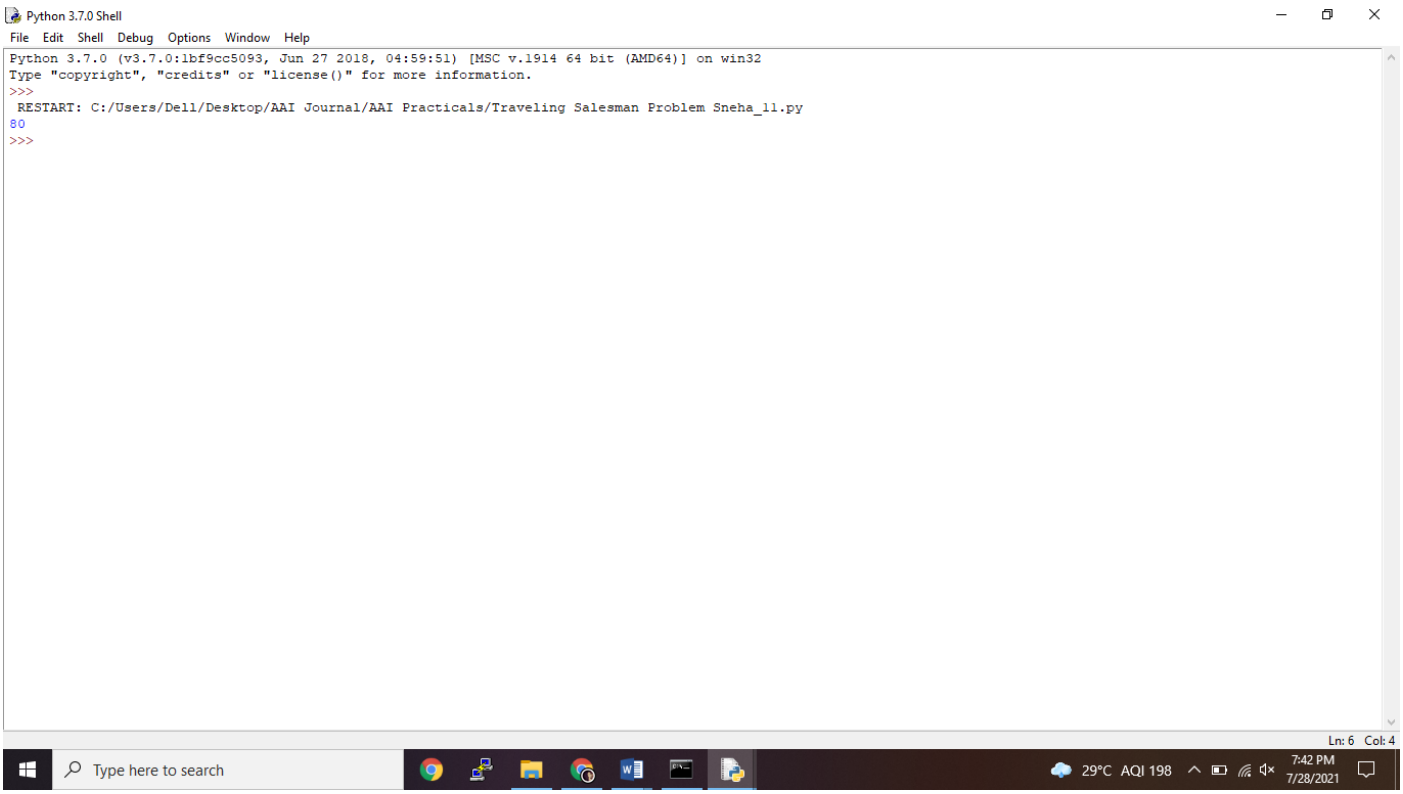
CODE:

```
# Python3 program to implement traveling salesman
# problem using naive approach.
from sys import maxsize
from itertools import permutations
V = 4
# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):
    # store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    # store minimum weight Hamiltonian Cycle
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
 # store current Path weight(cost)
        current_pathweight = 0
        # compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
```

```
        # update minimum
        min_path = min(min_path, current_pathweight)
    return min_path
# Driver Code
if __name__ == "__main__":

    # matrix representation of graph
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```



OUTPUT:



Drive Link for Code:

https://drive.google.com/file/d/1JYaqRJRzAfKVspULw6ASaIC6Iwe9II4V/view?usp=sharing

PRACTICAL - 8

Mutilated Checkerboard Problem

A covering of dominoes matches each white square with a black square, so we have two collections that we are trying to match. If some collection of white squares collectively are not attached to enough black squares, a covering is impossible. Therefore:

- To show that a mutilated chess board is coverable - cover it.
- To show that it is not coverable, demonstrate a collection of black squares which are collectively attached to insufficiently many white squares.
  - (or *vice versa*)

We can now use this result to create a minimal, connected, uncoverable set of squares. Here's how.

To be uncoverable we must have a collection of black squares that collectively are connected to insufficiently many white squares. If we used just one black square then it would have to be connected to no squares at all, and so it would be disconnected, so we must use at least two black squares. They must then be attached to just one white square.

Now we need at least one more white square to balance the numbers, but that can't be connected to either of the black squares we already have. That means we need another black square. We now have a white square with three black neighbours, and so we need ***two*** more white squares to balance the numbers. These must only be attached to one of the black squares, and there's only one way to do that.

And we're done! By construction, we have a provably minimal uncoverable configuration. If you like you can see how every possible configuration of four can be covered, and every other configuration of six can also be covered. This is the unique shape. It also lets us answer the question above about the mutilated chessboard with three of each colour missing.

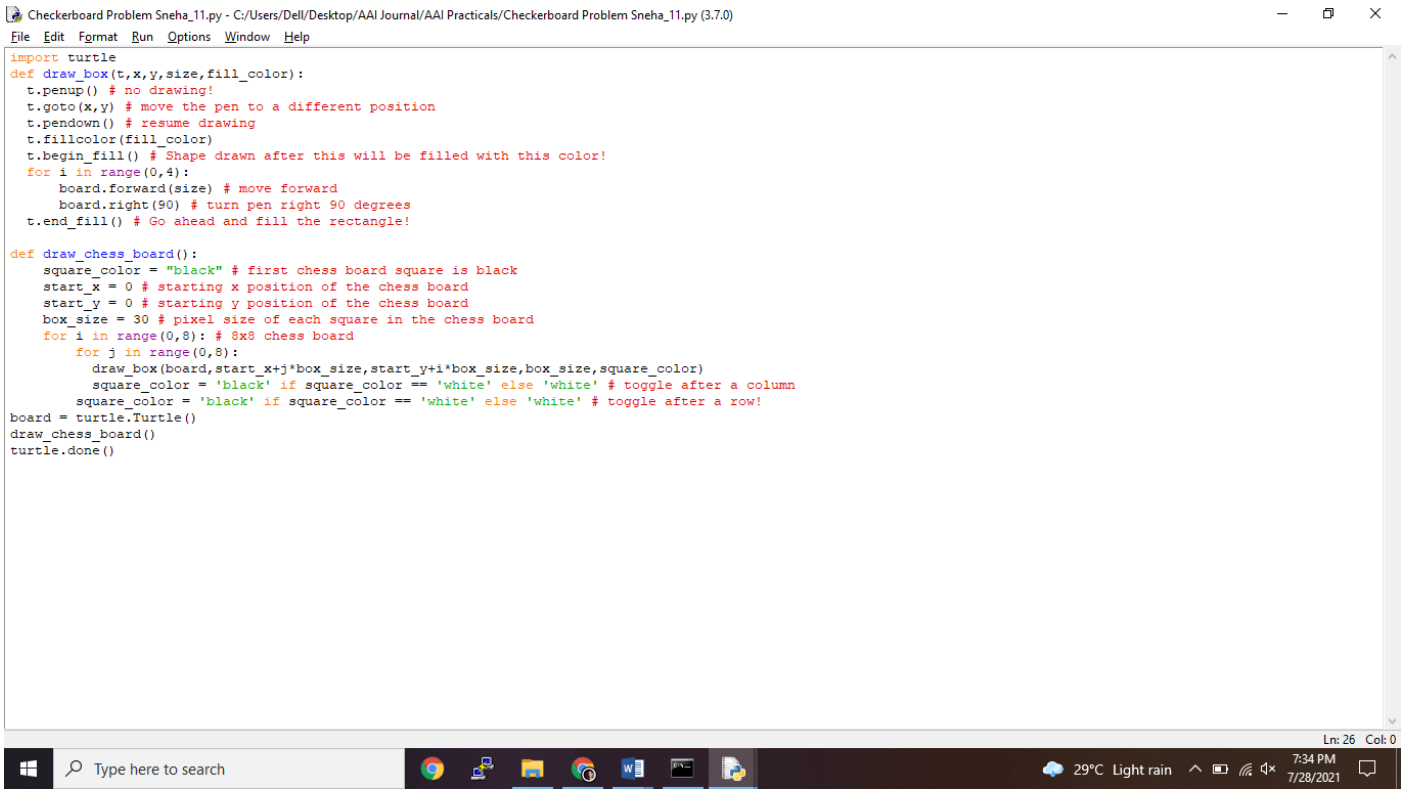
For simple drawing, python provides a built-in module named **turtle**. This behaves like a drawing board and you can use various drawing commands to draw over it. The basic commands control the movement of the drawing pen itself.

CODE:

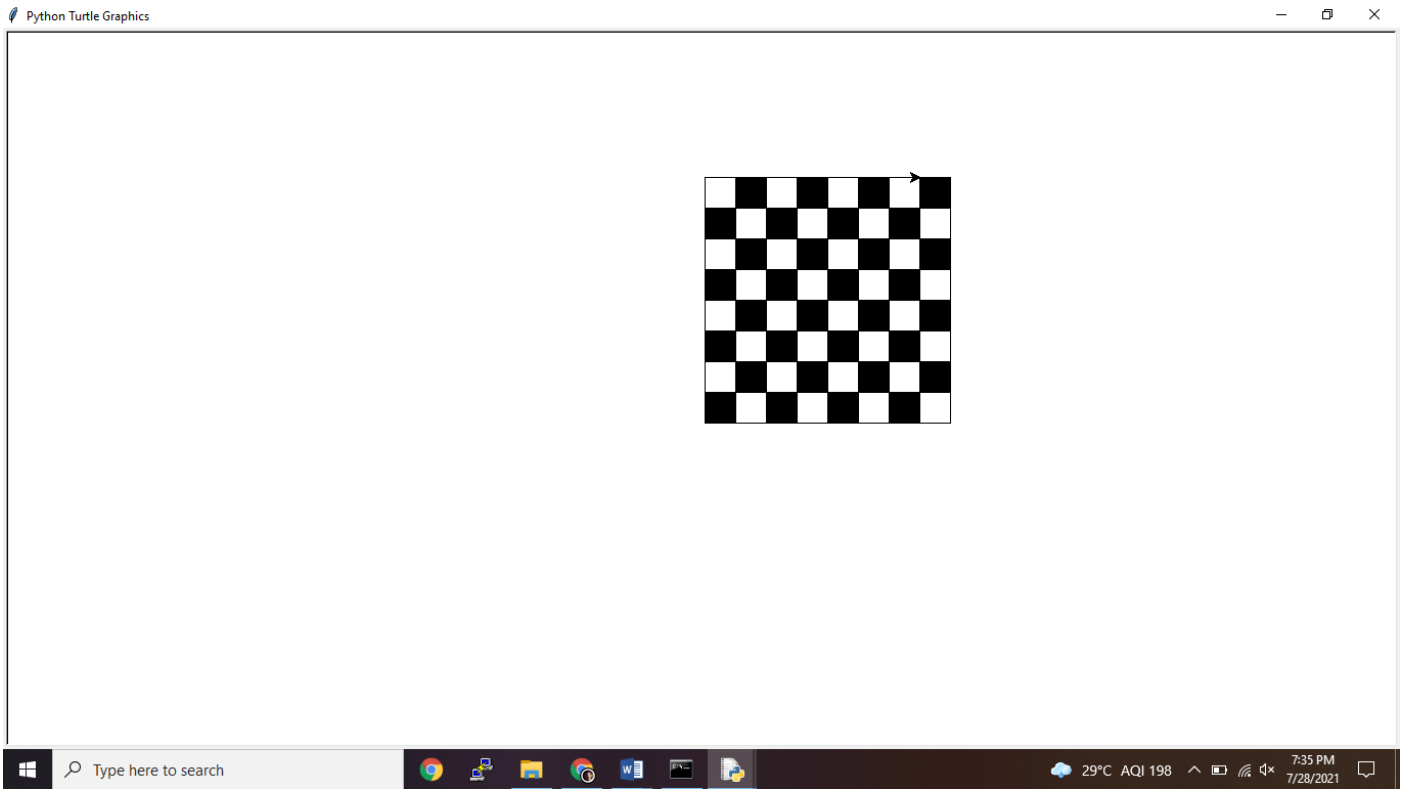
Checkerboard Problem.py :

```
import turtle
def draw_box(t,x,y,size,fill_color):
  t.penup() # no drawing!
  t.goto(x,y) # move the pen to a different position
  t.pendown() # resume drawing
  t.fillcolor(fill_color)
  t.begin_fill() # Shape drawn after this will be filled with this color!
  for i in range(0,4):
      board.forward(size) # move forward
      board.right(90) # turn pen right 90 degrees
  t.end_fill() # Go ahead and fill the rectangle!
```

```
def draw_chess_board():
    square_color = "black" # first chess board square is black
    start_x = 0 # starting x position of the chess board
    start_y = 0 # starting y position of the chess board
    box_size = 30 # pixel size of each square in the chess board
    for i in range(0,8): # 8x8 chess board
        for j in range(0,8):

draw_box(board,start_x+j*box_size,start_y+i*box_size,box_size,square_color)
          square_color = 'black' if square_color == 'white' else 'white' # toggle
after a column
        square_color = 'black' if square_color == 'white' else 'white' # toggle
after a row!
board = turtle.Turtle()
draw_chess_board()
turtle.done()
```



OUTPUT:



Drive Link for code:

https://drive.google.com/file/d/1We6RsG8WVuKT4tZDxQi35hVd8ySj2gsw/view?usp=sharing