Source Code:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('/kaggle/input/car-driving-risk-analysis/car driving risk analysis.csv')
df.head()
```

Out[2]:

	speed	risk
0	200	95
1	90	20
2	300	98
3	110	60
4	240	72

In [3]:
```
# only 3rd bracket,, two dimension for input means independent variable
x = df[['speed']]
# only 3rd bracket,, one dimension for output means dependent variable
y = df['risk']
```

In [4]:
```
x.head()
```

Out[4]:

	speed
0	200
1	90
2	300
3	110
4	240

In [5]:
```
y.head()
```

Out[5]:
```
0    95
1    20
2    98
3    60
4    72
Name: risk, dtype: int64
```

In [6]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.4, random_state=1)
```

In [7]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.4, random_state=1)
```

In [8]:
```
xtrain
```

Out[8]:

	speed
1	90
13	95
0	200
14	30
9	260
8	190
12	310
11	185
5	115

In [9]:
```
xtest
```

Out[9]:

	speed
3	110
7	230
6	50
2	300
10	290
4	240

In [10]:
```
ytrain
```

Out[10]:
```
1     20
13    18
0     95
14     2
9     91
8     45
12    93
11    59
5     10
Name: risk, dtype: int64
```

In [11]:
```
ytest
```

Out[11]:
```
3     60
7     85
6      7
2     98
10    82
4     72
Name: risk, dtype: int64
```

In [12]:
```
from sklearn.linear_model import LinearRegression
```

In [13]:
```
reg = LinearRegression()  # create object
```

In [14]:
```
reg.fit(xtrain, ytrain)
```

Out[14]:
```
LinearRegression()
```

In [15]:
```
reg.predict(xtest)
```

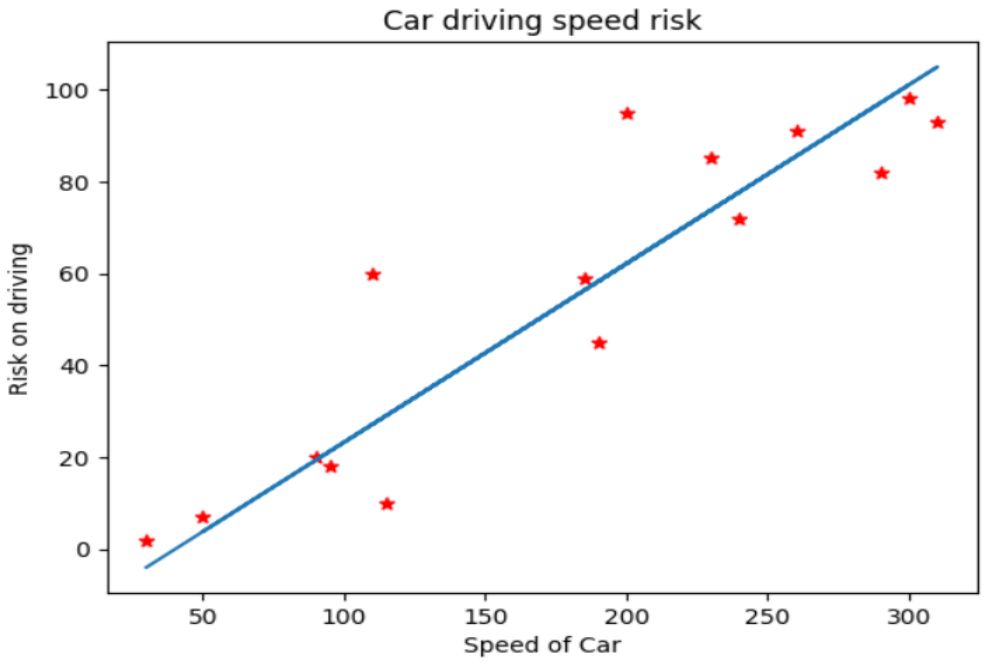
Out[15]:
```
array([ 27.15301215,  73.82259334,   3.81822156, 101.04651569,
        97.15738393,  77.7117251 ])
```

In [16]:
```
ytest
```

Out[16]:
```
3     60
7     85
6      7
2     98
10    82
4     72
Name: risk, dtype: int64
```

In [17]:
```
plt.scatter(df['speed'], df['risk'], marker='*', color='red')
plt.xlabel('Speed of Car')
plt.ylabel('Risk on driving')
plt.title('Car driving speed risk')
plt.plot(df.speed, reg.predict(df[['speed']]))
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x78a34087f940>]
```



In [18]:
```
reg.predict([[150]])
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does not have valid fea
ture names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Out[18]:
```
array([42.70953921])
```

In [19]:
```
reg.coef_
```

Out[19]:
```
array([0.38891318])
```

In [20]:
```
reg.intercept_
```

Out[20]:
```
-15.62743726501705
```

In [21]:
```
y=0.38891318*150-15.627437265017058

y
```

Out[21]:
```
42.70953973498295
```

Source Code:

```
def tower_of_hanoi(n, from_rod, to_rod, aux_rod):
    if n == 1:
        print(f"Move disk 1 from rod {from_rod} to rod {to_rod}")
        return
    tower_of_hanoi(n - 1, from_rod, aux_rod, to_rod)
    print(f"Move disk {n} from rod {from_rod} to rod {to_rod}")
    tower_of_hanoi(n - 1, aux_rod, to_rod, from_rod)

# Prompt the user to enter the number of disks
n = int(input("Enter the number of disks: "))

# A, B, and C are names of rods
tower_of_hanoi(n, 'A', 'C', 'B')
```

Output:

```
Enter the number of disks: 3
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

Source Code:

```
from collections import deque


def bfs(v):
    queue = deque([v])
    visited[v] = True
    while queue:
        node = queue.popleft()
        for i in range(1, n + 1):
            if adj_matrix[node][i] == 1 and not visited[i]:
                queue.append(i)
                visited[i] = True

# Input: number of vertices, adjacency matrix, and starting vertex
n = int(input("Enter number of vertices: "))
print("Enter adjacency matrix of the graph:")
adj_matrix = [[0] * (n + 1)] + [[0] + list(map(int, input().split())) for _ in range(n)]
visited = [False] * (n + 1)
v = int(input("Enter starting vertex: "))

# Perform BFS and output reachable nodes
bfs(v)
print("Reachable nodes:", " ".join(map(str, [i for i in range(1, n + 1) if visited[i]])))
```

Output:

```
Enter the number of vertices:5

Enter graph data in matrix form:
1 0 1 0 0
0 1 1 1 0
```

1 0 0 1 1

1 1 1 0 0

0 1 0 1 0

Enter the starting vertex:1

The node which are reachable are:

1 2 3 4 5

Source Code:

```
# Function to perform DFS on the graph
def dfs(node):
    print(node, end=" ")
    visited[node] = True
    for neighbor in range(n):
        if not visited[neighbor] and adj_matrix[node][neighbor] == 1:
            dfs(neighbor)

# Input number of vertices and adjacency matrix
n = int(input("Enter number of vertices: "))
print("Enter adjacency matrix of the graph:")
adj_matrix = [list(map(int, input().split())) for _ in range(n)]

# Initialize visited list and perform DFS
visited = [False] * n
print("DFS traversal starting from vertex 0:")
dfs(0)
```

Output:

```
Enter number of vertices: 5
Enter adjacency matrix of the graph:
1 0 1 0 0
0 1 1 1 0
1 0 0 1 1
1 1 1 0 0
0 1 0 1 0
DFS traversal starting from vertex 0:
0 2 3 1 4
```

Source Code:

```
def take_input():
    n = int(input("Enter the number of nodes: "))
    print("Enter the cost matrix:")
    cost_matrix = [list(map(int, input().split())) for _ in range(n)]
    return n, cost_matrix


def find_min_cost_path(city, n, cost_matrix, visited):
    print(f"{city + 1} ---> ", end="")
    visited[city] = True
    min_distance, next_city = float('inf'), -1

    # Find the nearest unvisited city
    for i in range(n):
        if cost_matrix[city][i] and not visited[i] and cost_matrix[city][i] < min_distance:
            min_distance, next_city = cost_matrix[city][i], i

    if next_city == -1:
        print("1")
        return cost_matrix[city][0]  # Return to starting city
    return min_distance + find_min_cost_path(next_city, n, cost_matrix, visited)

# Main program
n, cost_matrix = take_input()
print("\nThe Path is:")
total_cost = find_min_cost_path(0, n, cost_matrix, [False] * n)
print(f"\n\nMinimum cost is {total_cost}")
```

Output:

```
Enter the number of nodes: 4
Enter the cost matrix:
```

0 5 4 5

1 0 5 4

2 4 0 3

1 2 5 0

The Path is:

1 ---> 3 ---> 4 ---> 2 ---> 1

Minimum cost is 10

Source Code:

[9]:
```
import pandas as pd
data= {'Name': ['Mitu','Shahadat','Masud','Sourov','Mahi','Sumaiya'],
'Age': [23,23,24,22,22,20],
'Gender': ['F','M','M','M','F','F'],
'Marks': [80,83,'NaN',90,92,'NaN']}
df =pd.DataFrame(data)
df
```

[9]:

	Name	Age	Gender	Marks
0	Mitu	23	F	80
1	Shahadat	23	M	83
2	Masud	24	M	NaN
3	Sourov	22	M	90
4	Mahi	22	F	92
5	Sumaiya	20	F	NaN

[10]:
```
dataset = pd.read_csv('/kaggle/input/car-driving-risk-analysis/car driving risk analysis.csv')
dataset
```

[10]:

	speed	risk
0	200	95
1	90	20
2	300	98
3	110	60
4	240	72
5	115	10
6	50	7
7	230	85
8	190	45
9	260	91
10	290	82
11	185	59
12	310	93

```
import statistics as st
import pandas as pd

# Read the CSV file
df = pd.read_csv('/kaggle/input/car-driving-risk-analysis/car driving risk analysis.csv')

# Calculate and print statistical values
print("Mean:", st.mean(df['speed']))
print("Median:", st.median(df['speed']))
print("Mode:", st.mode(df['speed']))
print("Standard Deviation:", st.stdev(df['speed']))
print("Variance:", st.variance(df['speed']))
```

```
Mean: 179.66666666666666
Median: 190
Mode: 200
Standard Deviation: 92.68739062543719
Variance: 8590.952380952382
```

Source Code:

```
board = [' ' for _ in range(9)]


def print_board():
    for i in range(3):
        print("|".join(board[i*3:(i+1)*3]))
        if i < 2: print("-----")


def check_winner(player):
    win_cond = [(0,1,2), (3,4,5), (6,7,8), (0,3,6), (1,4,7), (2,5,8), (0,4,8), (2,4,6)]
    return any(board[a] == board[b] == board[c] == player for a, b, c in win_cond)


def play_game():
    player = 'X'
    for _ in range(9):
        print_board()
        move = int(input(f"Player {player}, choose (1-9): ")) - 1
        if board[move] == ' ':
            board[move] = player
            if check_winner(player):
                print_board()
                print(f"Player {player} wins!")
                return
            player = 'O' if player == 'X' else 'X'
        else:
            print("Invalid move, try again.")
    print_board()
    print("It's a draw!")


play_game()
```

O|X|X

O| |

 |O|X

Player X, choose (1-9): 6

O|X|X

O| |X

 |O|X

Player X wins!