DAA

1) Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

Code:

```
import timeit


def fibonacci(n):
    for i in range(2, n + 1):
        fib_list[i] = fib_list[i - 1] + fib_list[i - 2]
    return fib_list[n]


def fibonacci_recursive(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    fib_recur_list[n] = fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
    return fib_recur_list[n]


N = 20
RUNS = 1000
print(f"Given N = {N}\n{RUNS} runs")


fib_list = [0] * (N + 1)
fib_list[0] = 0
fib_list[1] = 1
print(
    "Fibonacci non-recursive:",
    fibonacci(N),
    "\tTime:",
    f'{timeit.timeit("fibonacci(N)", setup=f"from __main__ import fibonacci;N={N}", number=RUNS):5f}',
    "O(n)\tSpace: O(1)",
)
```

```
fib_recur_list = [0] * (N + 1)

fib_recur_list[0] = 0

fib_recur_list[1] = 1

print(

    "Fibonacci recursive:\t",

    fibonacci_recursive(N),

    "\tTime:",

    f'{timeit.timeit("fibonacci_recursive(N)", setup=f"from __main__ import fibonacci_recursive;N={N}",
number=RUNS,):5f}',

    "O(2^n)\tSpace: O(n)",

)
```

Output:

```
Given N = 20

1000 runs

Fibonacci non-recursive: 6765   Time: 0.001657 O(n)     Space: O(1)

Fibonacci recursive:     6765   Time: 2.064246 O(2^n)   Space: O(n)
```

2) Write a program to implement Huffman Encoding using a greedy strategy.

Code:

```
class Node:

    def __init__(self, freq_, symbol_, left_=None, right_=None):
        self.freq = freq_

        self.symbol = symbol_

        self.left = left_

        self.right = right_
        self.huff = ""

def print_nodes(node, val=""):
    # huffman code for current node
    new_val = val + str(node.huff)

    if node.left:
        print_nodes(node.left, new_val)
    if node.right:
        print_nodes(node.right, new_val)



    if not node.left and not node.right:
        print(f"{node.symbol} -> {new_val}")

chars = ["a", "b", "c", "d", "e", "f"]

# frequency of characters
freq = [5, 9, 12, 13, 16, 45]

nodes = [Node(freq[x], chars[x]) for x in range(len(chars))]
```

```
while len(nodes) > 1:
    # sort all the nodes in ascending order based on their frequency
    nodes = sorted(nodes, key=lambda x: x.freq)



    left = nodes[0]
    right = nodes[1]
    left.huff = 0
    right.huff = 1
    newNode = Node(left.freq + right.freq, left.symbol + right.symbol, left, right)

    nodes.remove(left)
    nodes.remove(right)
    nodes.append(newNode)

print("Characters :", f'[{", ".join(chars)}]')
print("Frequency  :", freq, "\n\nHuffman Encoding:")
print_nodes(nodes[0])
```

Output:

```
Characters : [a, b, c, d, e, f]
Frequency  : [5, 9, 12, 13, 16, 45]
Huffman Encoding:
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
```

3) Write a program to solve a fractional Knapsack problem using a greedy method.

Code:

```
class ItemValue:
    """Item Value DataClass"""

    def __init__(self, wt_, val_, ind_):
        self.wt = wt_
        self.val = val_
        self.ind = ind_
        self.cost = val_ // wt_

    def __lt__(self, other):
        return self.cost < other.cost



def fractionalKnapSack(wt, val, capacity):
    """Function to get maximum value"""
    iVal = [ItemValue(wt[i], val[i], i) for i in range(len(wt))]

    # sorting items by cost
    iVal.sort(key=lambda x: x.cost, reverse=True)

    totalValue = 0
    for i in iVal:
        curWt = i.wt
        curVal = i.val
        if capacity - curWt >= 0:
            capacity -= curWt
            totalValue += curVal
        else:
            fraction = capacity / curWt
            totalValue += curVal * fraction
            capacity = int(capacity - (curWt * fraction))
```

```
        break
    return totalValue



if __name__ == "__main__":
    wt = [10, 40, 20, 30]
    val = [60, 40, 100, 120]
    capacity = 50

    # Function call
    maxValue = fractionalKnapSack(wt, val, capacity)
    print("Maximum value in Knapsack =", maxValue)
```

Output:

Maximum value in Knapsack = 240.0

4) Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy

Output:

```
def knapsack_dp(W, wt, val, n):
    """A Dynamic Programming based solution for 0-1 Knapsack problem
    Returns the maximum value that can"""
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    # Build table K[][] in bottom up manner
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w])
            else:
                K[i][w] = K[i - 1][w]
    return K[n][W]



val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print("Maximum possible profit =", knapsack_dp(W, wt, val, n))
```

Output:

Maximum possible profit = 220

5) Design n-Queens matrix having first Queen placed. Use backtracking to place remaining

Queens to generate the final n-queen's matrix

Code:

```
class NQBacktracking:
   def __init__(self, x_, y_):
      self.ld = [0] * 30
      self.rd = [0] * 30
      self.cl = [0] * 30
      self.x = x_
      self.y = y_

   def printSolution(self, board):
      print(
         "N Queen Backtracking Solution:\nGiven initial position of 1st queen at row:",
         self.x,
         "column:",
         self.y,
         "\n",
      )
      for line in board:
         print(" ".join(map(str, line))

   def solveNQUtil(self, board, col):
      if col >= N:
         return True
      if col == self.y:
         return self.solveNQUtil(board, col + 1)
      for i in range(N):
         if i == self.x:
            continue
         if (self.ld[i - col + N - 1] != 1 and self.rd[i + col] != 1) and self.cl[i] != 1:
            board[i][col] = 1
            self.ld[i - col + N - 1] = self.rd[i + col] = self.cl[i] = 1
```

```
            if self.solveNQUtil(board, col + 1):
                return True
            board[i][col] = 0
            self.ld[i - col + N - 1] = self.rd[i + col] = self.cl[i] = 0
        return False

    def solveNQ(self):
        board = [[0 for _ in range(N)] for _ in range(N)]
        board[self.x][self.y] = 1
        self.ld[self.x - self.y + N - 1] = self.rd[self.x + self.y] = self.cl[self.x] = 1
        if not self.solveNQUtil(board, 0):
            print("Solution does not exist")
            return False
        self.printSolution(board)
        return True

if __name__ == "__main__":
    N = 8
    x, y = 3, 2
    NQBt = NQBacktracking(x, y)
    NQBt.solveNQ()
```

OUTPUT:

```
N Queen Backtracking Solution:
Given initial position of 1st queen at row: 3 column: 2

1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
```

In [4]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing
```

In [5]:

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

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         200000 non-null  int64
 1   key                200000 non-null  object
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [7]:

```
df.head()
```

Out[7]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40

In [8]:

```
df.describe()
```

Out[8]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

In [9]:

```
df = df.drop(['Unnamed: 0', 'key'], axis=1)
```

In [10]:

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

Out[10]:

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [11]:

```
df.dropna(axis=0,inplace=True)
```

In [12]:

```
df.dtypes
```

Out[12]:

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [13]:

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```

In [14]:

```
df= df.assign(
    second = df.pickup_datetime.dt.second,
    minute = df.pickup_datetime.dt.minute,
    hour = df.pickup_datetime.dt.hour,
    day= df.pickup_datetime.dt.day,
    month = df.pickup_datetime.dt.month,
    year = df.pickup_datetime.dt.year,
    dayofweek = df.pickup_datetime.dt.dayofweek
)
df = df.drop('pickup_datetime',axis=1)
```

In [15]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199999 entries, 0 to 199999
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        199999 non-null  float64
 1   pickup_longitude   199999 non-null  float64
 2   pickup_latitude    199999 non-null  float64
 3   dropoff_longitude  199999 non-null  float64
 4   dropoff_latitude   199999 non-null  float64
 5   passenger_count    199999 non-null  int64  
 6   second             199999 non-null  int64  
 7   minute             199999 non-null  int64  
 8   hour               199999 non-null  int64  
 9   day                199999 non-null  int64  
 10  month              199999 non-null  int64  
 11  year               199999 non-null  int64  
 12  dayofweek          199999 non-null  int64  
dtypes: float64(5), int64(8)
memory usage: 21.4 MB
```

In [16]:

```
df.head()
```

Out[16]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	second	minute	hour
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1	6	52	19
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1	56	4	20
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1	0	45	21
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3	21	22	8
4	16.0	-73.925023	40.744085	-73.973082	40.761247	5	0	47	17

In [17]:

```
incorrect_coordinates = df.loc[
    (df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
    (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
    (df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
    (df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

In [18]:

```
def distance_transform(longitude1, latitude1, longitude2, latitude2):
    long1, lati1, long2, lati2 = map(np.radians, [longitude1, latitude1, longitude2, lat
itude2])
    dist_long = long2 - long1
    dist_lati = lati2 - lati1
    a = np.sin(dist_lati/2)**2 + np.cos(lati1) * np.cos(lati2) * np.sin(dist_long/2)**2
    c = 2 * np.arcsin(np.sqrt(a)) * 6371
    # long1,lati1,long2,lati2 = longitude1[pos],latitude1[pos],longitude2[pos],latitude2[
pos]
    # c = sqrt((long2 - long1) ** 2 + (lati2 - lati1) ** 2)asin
       
    return c
```

In [19]:

```
df['Distance'] = distance_transform(
```

```
    df['pickup_longitude'],
    df['pickup_latitude'],
    df['dropoff_longitude'],
    df['dropoff_latitude']
)
```

In [20]:

```
df.head()
```

Out[20]:

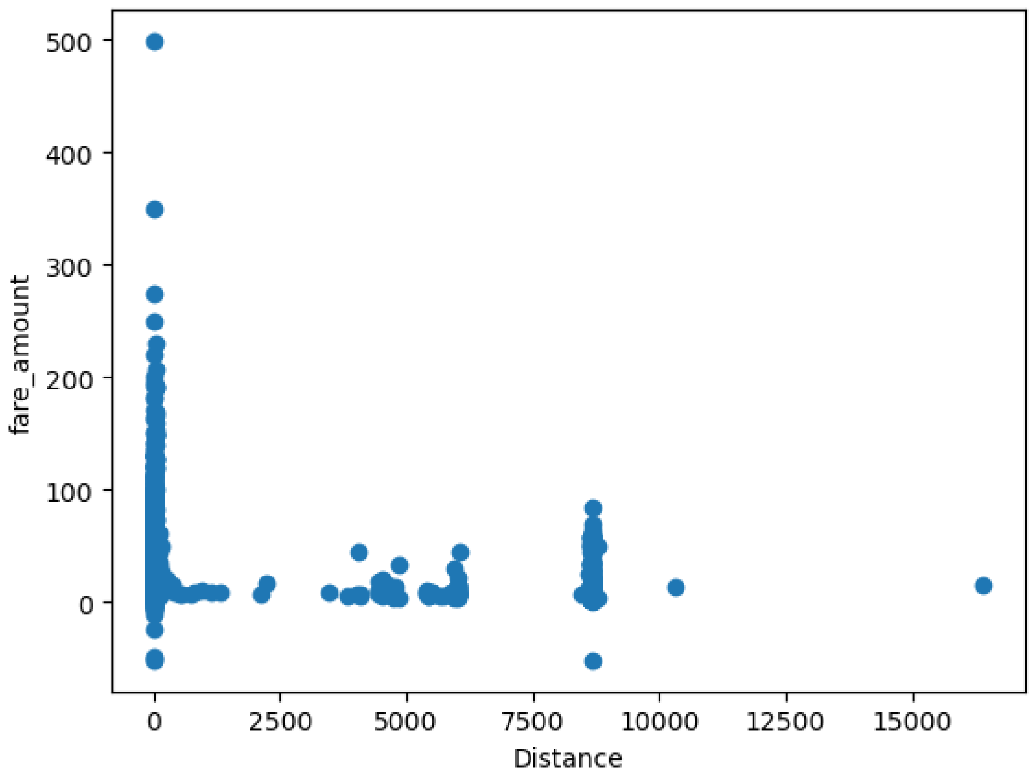
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	second	minute	hour
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1	6	52	19
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1	56	4	20
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1	0	45	21
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3	21	22	8
4	16.0	-73.925023	40.744085	-73.973082	40.761247	5	0	47	17

In [21]:

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

Out[21]:

```
Text(0, 0.5, 'fare_amount')
```

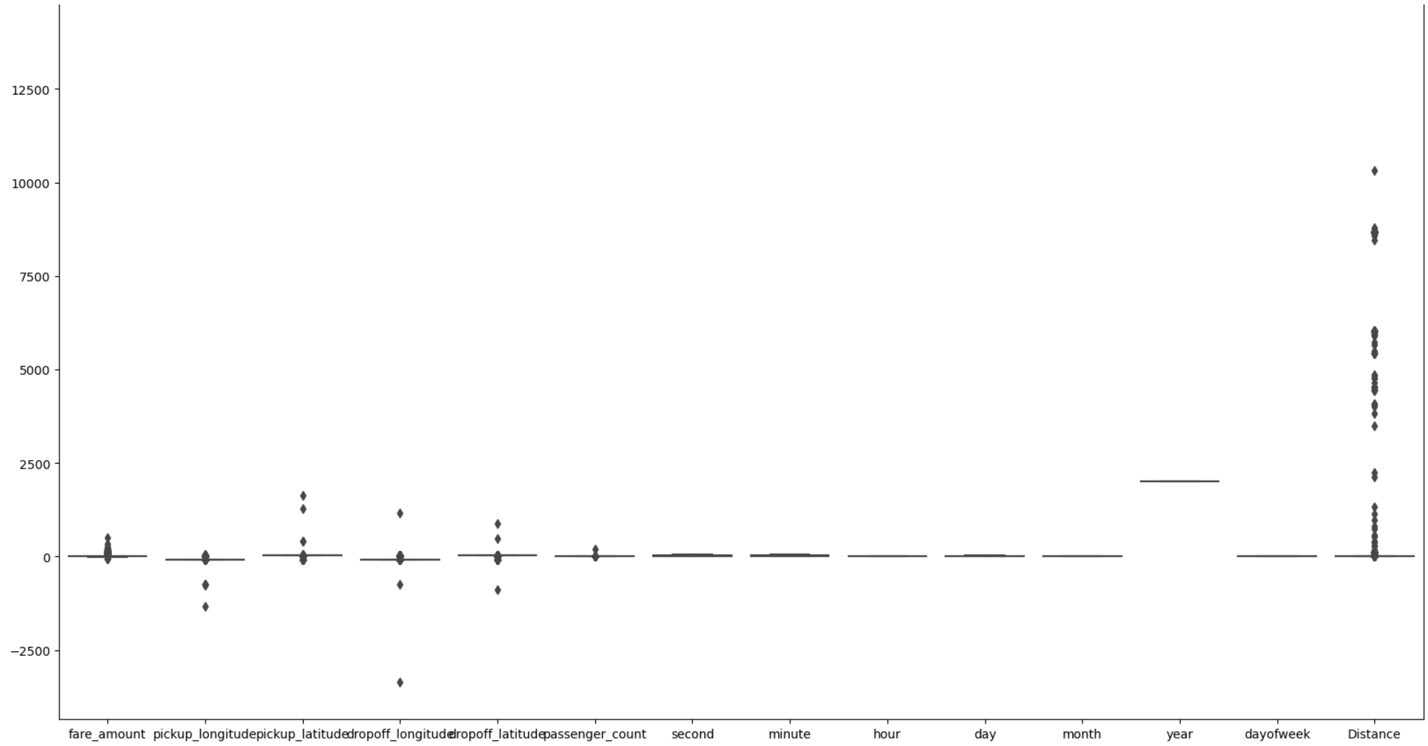


In [22]:

```
plt.figure(figsize=(20,12))
sns.boxplot(data = df)
```

Out[22]:

```
<Axes: >
```



In [23]:

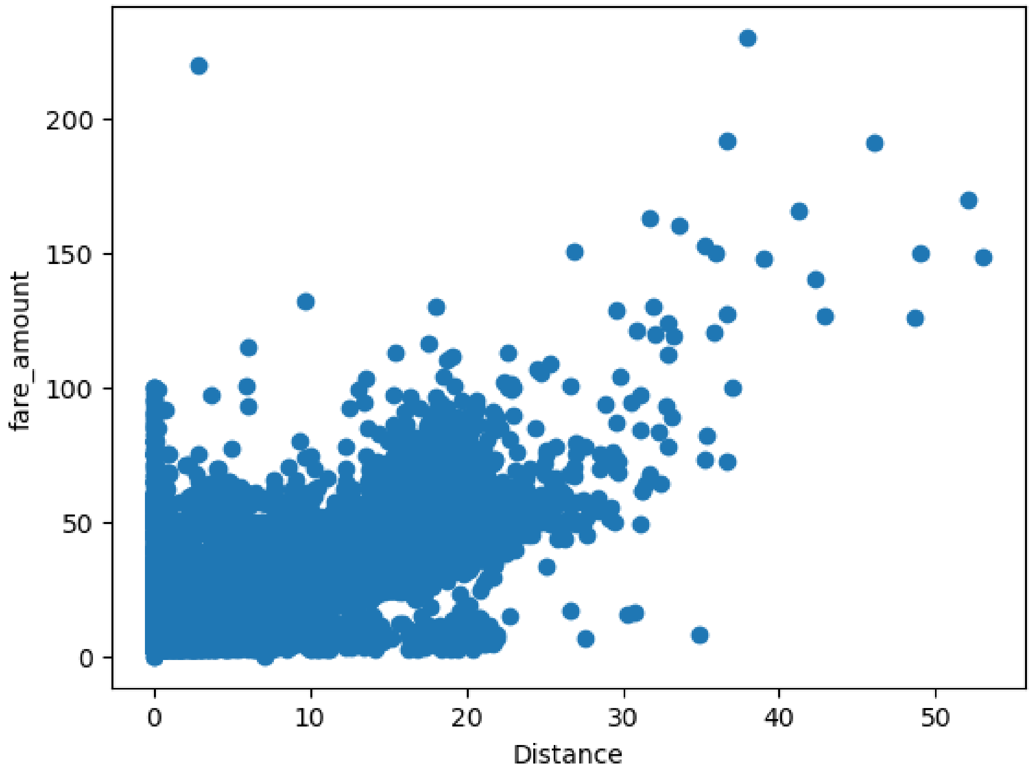
```
df.drop(df[df['Distance'] >= 60].index, inplace = True)
df.drop(df[df['fare_amount'] <= 0].index, inplace = True)

df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace = True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index, inplace = True )
```

In [24]:

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

Out[24]:

```
Text(0, 0.5, 'fare_amount')
```



In [25]:

```
corr = df.corr()

corr.style.background_gradient(cmap='BuGn')
```

Out[25]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	seco
fare_amount	1.000000	0.005885	-0.006253	0.005501	-0.006142	0.011693	0.0009
pickup_longitude	0.005885	1.000000	-0.973204	0.999992	-0.981941	-0.000649	0.0146
pickup_latitude	-0.006253	-0.973204	1.000000	-0.973206	0.991076	-0.001190	0.0168
dropoff_longitude	0.005501	0.999992	-0.973206	1.000000	-0.981942	-0.000650	0.0146
dropoff_latitude	-0.006142	-0.981941	0.991076	-0.981942	1.000000	-0.001035	0.0172
passenger_count	0.011693	-0.000649	-0.001190	-0.000650	-0.001035	1.000000	0.2029
second	-0.000995	-0.014677	0.016809	-0.014638	0.017202	-0.202987	1.0000
minute	-0.007795	0.002796	-0.002295	0.002803	-0.002593	0.000733	0.0018
hour	-0.020692	0.001547	-0.001823	0.001316	-0.001460	0.013226	0.0134
day	0.001059	0.005300	-0.008901	0.005307	-0.008900	0.003146	0.0021
month	0.023759	-0.002667	0.004098	-0.002656	0.004143	0.009921	0.0497
year	0.121195	0.005907	-0.008466	0.005878	-0.008553	0.004841	0.0831
dayofweek	0.006181	0.003006	-0.004787	0.003082	-0.004648	0.033360	0.0001
Distance	0.857729	-0.117044	0.110843	-0.117282	0.109486	0.007784	0.0003

In [26]:

```
X = df['Distance'].values.reshape(-1, 1)        #Independent Variable
y = df['fare_amount'].values.reshape(-1, 1)     #Dependent Variable
```

In [27]:

```
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(y)
print(y_std)

x_std = std.fit_transform(X)
print(x_std)
```

```
[[-0.39820843]
 [-0.37738556]
 [ 0.1640092 ]
 ...
 [ 2.03806797]
 [ 0.3305922 ]
 [ 0.28894645]]
[[-0.43819769]
 [-0.22258873]
 [ 0.49552213]
 ...
 [ 2.67145829]
```

```
 [ 0.07874908]
 [ 0.60173174]]
```

In [28]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2, random_
state=0)
```

In [29]:

```
from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

```
Training set score: 0.74
Test set score: 0.7340468
```

In [30]:

```
y_pred = l_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result[['Predicted']] = y_pred

result.sample(10)
```

Out[30]:

	Actual	Predicted
7830	-0.335740	-0.142014
6984	-0.710552	-0.585478
13803	0.039072	-0.245814
14761	0.039072	0.141870
33471	-0.335740	0.018925
33248	-0.398208	-0.380167
14803	-0.658494	-0.684020
23019	-0.210803	-0.425896
724	-0.502323	-0.229479
8576	4.573253	0.013334

In [31]:

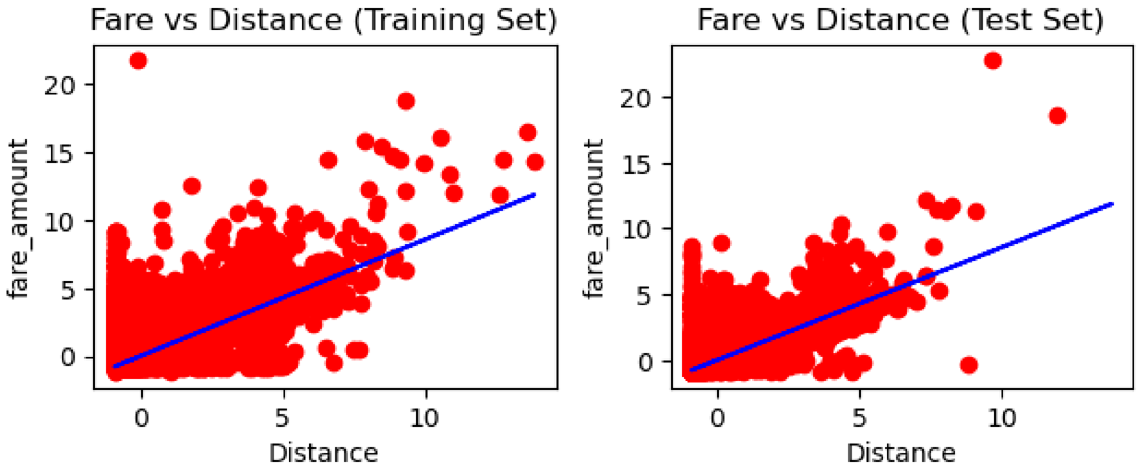
```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred)))
```

```
Mean Absolute Error: 0.2662129875793893
Mean Absolute % Error: 1.9830747633407433
Mean Squared Error: 0.27052435107785416
Root Mean Squared Error: 0.5201195546005304
R Squared (R²): 0.8567653080822022
```

In [32]:

```
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
```

```
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")

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



In [33]:

```
cols = ['Model', 'RMSE', 'R-Squared']

# create a empty dataframe of the colums
# columns: specifies the columns to be selected
result_tabulation = pd.DataFrame(columns = cols)

# compile the required information
linreg_metrics = pd.DataFrame([[
     "Linear Regresion model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
     np.sqrt(metrics.r2_score(y_test, y_pred))
]], columns = cols)

result_tabulation = pd.concat([result_tabulation, linreg_metrics], ignore_index=True)

result_tabulation
```

Out[33]:

	Model	RMSE	R-Squared
0	Linear Regresion model	0.52012	0.856765

In [ ]:

```
rf_reg = RandomForestRegressor(n_estimators=100, random_state=10)

# fit the regressor with training dataset
rf_reg.fit(X_train, y_train)
```

In [ ]:

```
# predict the values on test dataset using predict()
y_pred_RF = rf_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result['Predicted'] = y_pred_RF
```

```
result.sample(10)
```

In [ ]:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_RF))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred_RF
))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_RF))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred_RF)))
```

In [ ]:

```
# Build scatterplot
plt.scatter(X_test, y_test, c = 'b', alpha = 0.5, marker = '.', label = 'Real')
plt.scatter(X_test, y_pred_RF, c = 'r', alpha = 0.5, marker = '.', label = 'Predicted')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.grid(color = '#D3D3D3', linestyle = 'solid')
plt.legend(loc = 'lower right')


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

In [ ]:

```
# compile the required information
random_forest_metrics = pd.DataFrame([[
     "Random Forest Regressor model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)),
     np.sqrt(metrics.r2_score(y_test, y_pred_RF))
]], columns = cols)

result_tabulation = pd.concat([result_tabulation, random_forest_metrics], ignore_index=T
rue)

result_tabulation
```

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn import preprocessing
```

In [2]:

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

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
```

In [4]:

```
df.head()
```

Out[4]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	P
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	

5 rows × 3002 columns

In [5]:

```
df.dtypes
```

Out[5]:

```
Email No.      object
the             int64
to              int64
ect             int64
and             int64
                ...
military        int64
allowing        int64
ff              int64
dry             int64
Prediction      int64
Length: 3002, dtype: object
```

```
Length: 3002, dtype: object
```

In [6]:

```
df.drop(columns=['Email No.'], inplace=True)
```

In [7]:

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

Out[7]:

```
the           0
to            0
ect           0
and           0
for           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3001, dtype: int64
```

In [8]:

```
df.describe()
```

Out[8]:

	the	to	ect	and	for	of	a	you	hou
count	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000
mean	6.640565	6.188128	5.143852	3.075599	3.124710	2.627030	55.517401	2.466551	2.024362
std	11.745009	9.534576	14.101142	6.045970	4.680522	6.229845	87.574172	4.314444	6.967878
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	12.000000	0.000000	0.000000
50%	3.000000	3.000000	1.000000	1.000000	2.000000	1.000000	28.000000	1.000000	0.000000
75%	8.000000	7.000000	4.000000	3.000000	4.000000	2.000000	62.250000	3.000000	1.000000
max	210.000000	132.000000	344.000000	89.000000	47.000000	77.000000	1898.000000	70.000000	167.000000

8 rows × 3001 columns

In [9]:

```
X=df.iloc[:, :df.shape[1]-1]        #Independent Variables
y=df.iloc[:, -1]                    #Dependent Variable
X.shape, y.shape
```

Out[9]:

```
((5172, 3000), (5172,))
```

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=8
)
```

In [11]:

```
models = {
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=2),
    "Linear SVM":LinearSVC(random_state=8, max_iter=900000),
    "Polynomical SVM":SVC(kernel="poly", degree=2, random_state=8),
    "RBF SVM":SVC(kernel="rbf", random_state=8),
```

```
    "Sigmoid SVM":SVC(kernel="sigmoid", random_state=8)
}
```

In [12]:

```
for model_name, model in models.items():
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for {model_name} model \t: {metrics.accuracy_score(y_test, y_pred)}
")
```

```
Accuracy for K-Nearest Neighbors model  : 0.8878865979381443
Accuracy for Linear SVM model  : 0.9755154639175257
Accuracy for Polynomical SVM model  : 0.7615979381443299
Accuracy for RBF SVM model  : 0.8182989690721649
Accuracy for Sigmoid SVM model  : 0.6237113402061856
```

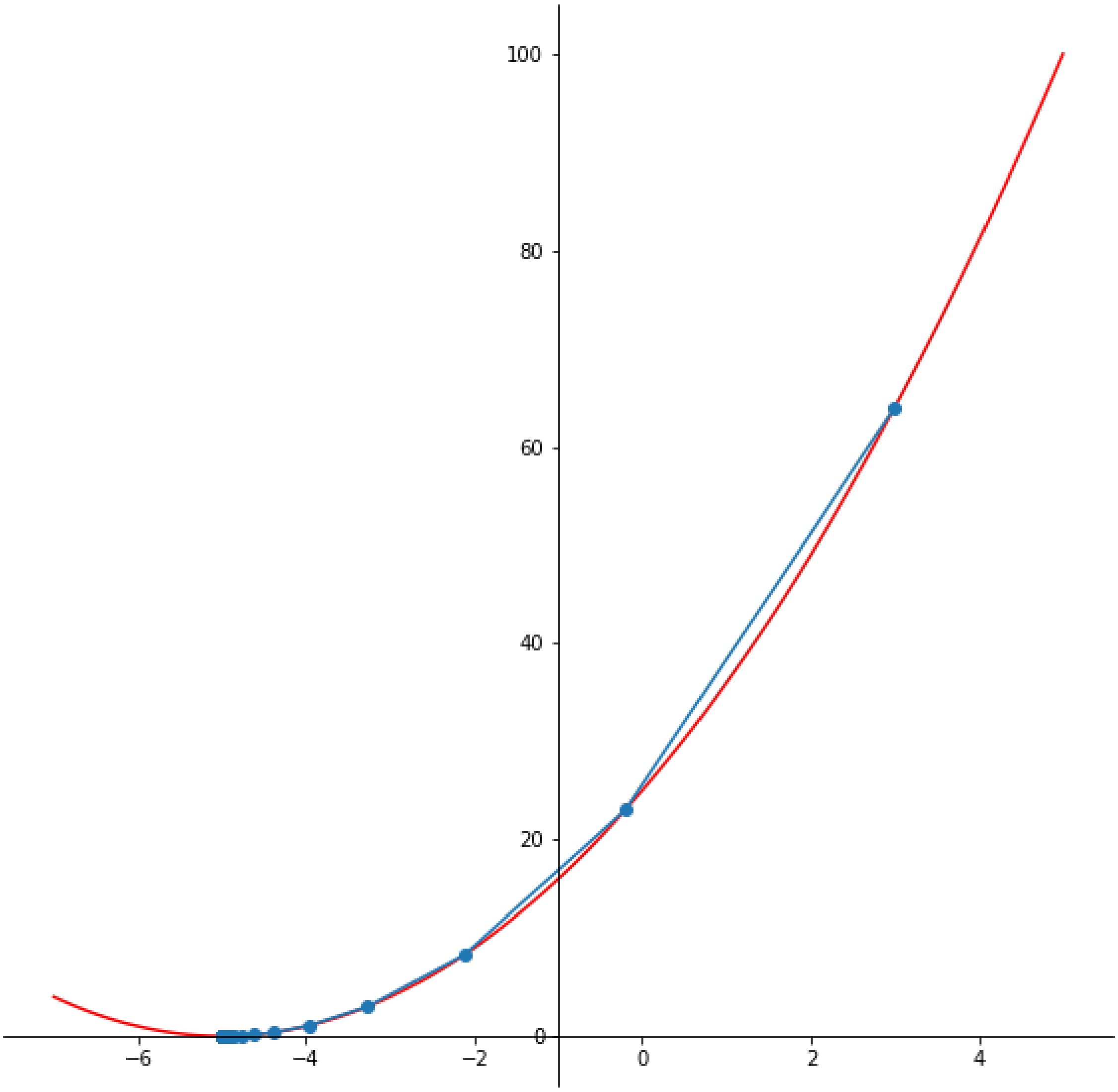
In [1]:
```
from sympy import Symbol, lambdify
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
x = Symbol('x')
```

In [3]:
```
def gradient_descent(
    function, start, learn_rate, n_iter=10000, tolerance=1e-06, step_size=1
):
    gradient = lambdify(x, function.diff(x))
    function = lambdify(x, function)
    points = [start]
    iters = 0                          #iteration counter

    while step_size > tolerance and iters < n_iter:
        prev_x = start                 #Store current x value in prev_x
        start = start - learn_rate * gradient(prev_x) #Grad descent
        step_size = abs(start - prev_x) #Change in x
        iters = iters+1                #iteration count
        points.append(start)
    print("The local minimum occurs at", start)

    # Create plotting array
    x_ = np.linspace(-7,5,100)
    y = function(x_)

    # setting the axes at the centre
    fig = plt.figure(figsize = (10, 10))
    ax = fig.add_subplot(1, 1, 1)
    ax.spines['left'].set_position('center')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.yaxis.set_ticks_position('left')

    # plot the function
    plt.plot(x_,y, 'r')
    plt.plot(points, function(np.array(points)), '-o')

    # show the plot
    plt.show()
```

In [4]:
```python
function=(x+5)**2

gradient_descent(
     function=function, start=3.0, learn_rate=0.2, n_iter=50
)
```

```
The local minimum occurs at -4.999998938845185
```



In [27]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn import preprocessing
```

Loading the Dataset

First we load the dataset and find out the number of columns, rows, NULL values, etc.

In [2]:
```python
df = pd.read_csv('diabetes.csv')
```

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Pregnancies    768 non-null    int64  
 1   Glucose        768 non-null    int64  
 2   BloodPressure  768 non-null    int64  
 3   SkinThickness  768 non-null    int64  
 4   Insulin        768 non-null    int64  
 5   BMI            768 non-null    float64
 6   Pedigree       768 non-null    float64
 7   Age            768 non-null    int64  
 8   Outcome        768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [4]:
```python
df.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Cleaning

In [11]:
```
df.corr().style.background_gradient(cmap='BuGn')
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647
Pedigree	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844

In [13]:
```
df.drop(['BloodPressure', 'SkinThickness'], axis=1, inplace=True)
```

In [14]:
```
df.isna().sum()
```

Out[14]:
```
Pregnancies    0
Glucose        0
Insulin        0
BMI            0
Pedigree       0
Age            0
Outcome        0
dtype: int64
```

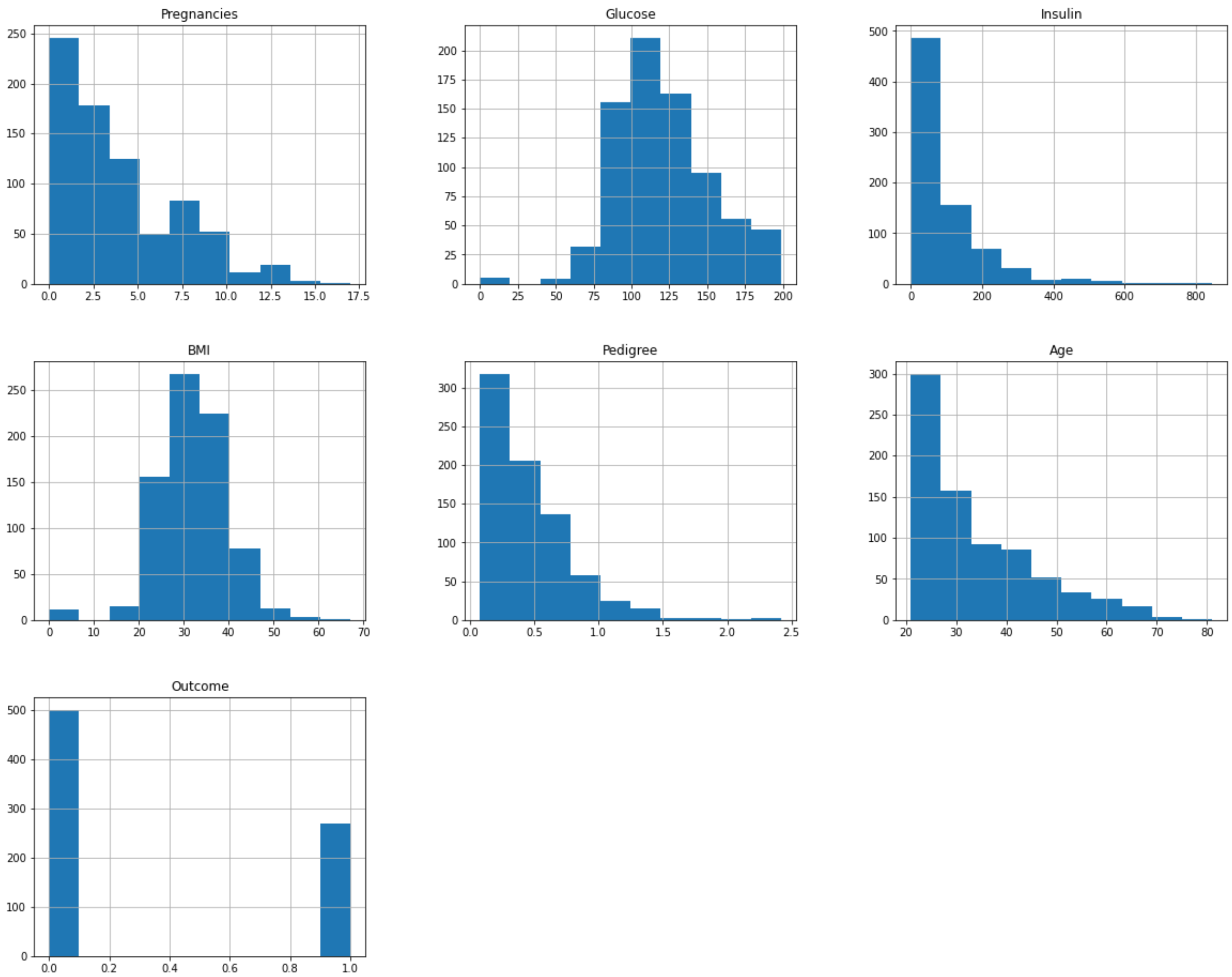
In [15]:
```
df.describe()
```

Out[15]:

	Pregnancies	Glucose	Insulin	BMI	Pedigree	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Visualization

In [16]:
```
hist = df.hist(figsize=(20,16))
```



Separating the features and the labels

In [17]:
```
X=df.iloc[:, :df.shape[1]-1]       #Independent Variables
y=df.iloc[:, -1]                   #Dependent Variable
X.shape, y.shape
```

Out[17]:
```
((768, 6), (768,))
```

Splitting the Dataset

Training and Test Set

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Machine Learning model

In [30]:
```
def knn(X_train, X_test, y_train, y_test, neighbors, power):
    model = KNeighborsClassifier(n_neighbors=neighbors, p=power)
    # Fit and predict on model
    # Model is trained using the train set and predictions are made based on the test s
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for K-Nearest Neighbors model \t: {accuracy_score(y_test, y_pred)}

    cm = confusion_matrix(y_test, y_pred)
    print(f'''Confusion matrix :\n
    | Positive Prediction\t| Negative Prediction
    ---------------+------------------------+----------------------
    Positive Class | True Positive (TP) {cm[0, 0]}\t| False Negative (FN) {cm[0, 1]}
    ---------------+------------------------+----------------------
    Negative Class | False Positive (FP) {cm[1, 0]}\t| True Negative (TN) {cm[1, 1]}\n'
    cr = classification_report(y_test, y_pred)
    print('Classification report : \n', cr)
```

Hyperparameter tuning

In [28]:
```
param_grid = {
    'n_neighbors': range(1, 51),
    'p': range(1, 4)
}
grid = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
grid.best_estimator_, grid.best_params_, grid.best_score_
```

Out[28]:
```
(KNeighborsClassifier(n_neighbors=27),
 {'n_neighbors': 27, 'p': 2},
 0.7719845395175262)
```

In [31]:
```
knn(X_train, X_test, y_train, y_test, grid.best_params_['n_neighbors'], grid.best_param
```

```
Accuracy for K-Nearest Neighbors model  : 0.7987012987012987
Confusion matrix :

    | Positive Prediction       | Negative Prediction
    ---------------+------------------------+----------------------
    Positive Class | True Positive (TP) 91      | False Negative (FN) 11
    ---------------+------------------------+----------------------
    Negative Class | False Positive (FP) 20     | True Negative (TN) 32

Classification report :
               precision    recall  f1-score   support

           0       0.82      0.89      0.85       102
           1       0.74      0.62      0.67        52

    accuracy                           0.80       154
   macro avg       0.78      0.75      0.76       154
weighted avg       0.79      0.80      0.79       154
```

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

In [2]:
```python
df = pd.read_csv('sales_data_sample.csv', encoding='latin1')
df.head()
```

Out[2]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QT
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	

5 rows × 25 columns

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  ADDRESSLINE2      302 non-null    object 
 17  CITY              2823 non-null   object 
 18  STATE             1337 non-null   object 
 19  POSTALCODE        2747 non-null   object 
 20  COUNTRY           2823 non-null   object 
 21  TERRITORY         1749 non-null   object 
 22  CONTACTLASTNAME   2823 non-null   object 
 23  CONTACTFIRSTNAME  2823 non-null   object 
 24  DEALSIZE          2823 non-null   object 
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [4]:
```python
df.describe()
```

Out[4]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	M
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	282
mean	10258.725115	35.092809	83.658544	6.466171	3553.889072	2.717676	
std	92.085478	9.741443	20.174277	4.225841	1841.865106	1.203878	
min	10100.000000	6.000000	26.880000	1.000000	482.130000	1.000000	
25%	10180.000000	27.000000	68.860000	3.000000	2203.430000	2.000000	
50%	10262.000000	35.000000	95.700000	6.000000	3184.800000	3.000000	
75%	10333.500000	43.000000	100.000000	9.000000	4508.000000	4.000000	1
max	10425.000000	97.000000	100.000000	18.000000	14082.800000	4.000000	1

In [5]:
```python
fig = plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), annot=True, fmt='.2f')
plt.show()
```

In [6]:
```python
df= df[['PRICEEACH', 'MSRP']]
```

In [7]:
```python
df.head()
```

Out[7]:

	PRICEEACH	MSRP
0	95.70	95
1	81.35	95
2	94.74	95
3	83.26	95
4	100.00	95

In [8]:
```
df.isna().any()
```

Out[8]:
```
PRICEEACH    False
MSRP         False
dtype: bool
```

In [9]:
```
df.describe().T
```

Out[9]:

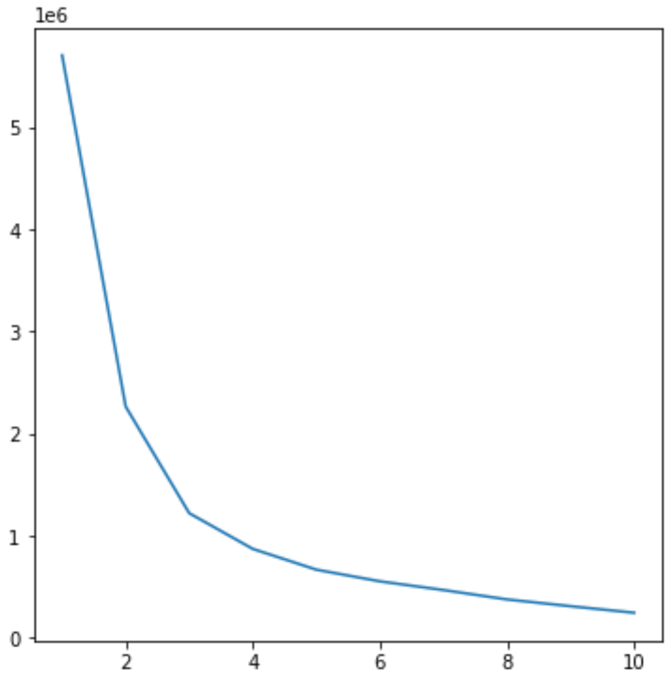
	count	mean	std	min	25%	50%	75%	max
PRICEEACH	2823.0	83.658544	20.174277	26.88	68.86	95.7	100.0	100.0
MSRP	2823.0	100.715551	40.187912	33.00	68.00	99.0	124.0	214.0

In [10]:
```
df.shape
```

Out[10]:
```
(2823, 2)
```

In [11]:
```
from sklearn.cluster import KMeans

inertia = []

for i in range(1, 11):
    clusters = KMeans(n_clusters=i, init='k-means++', random_state=42)
    clusters.fit(df)
    inertia.append(clusters.inertia_)

plt.figure(figsize=(6, 6))
sns.lineplot(x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], y = inertia)
```

Out[11]:
```
<AxesSubplot:>
```

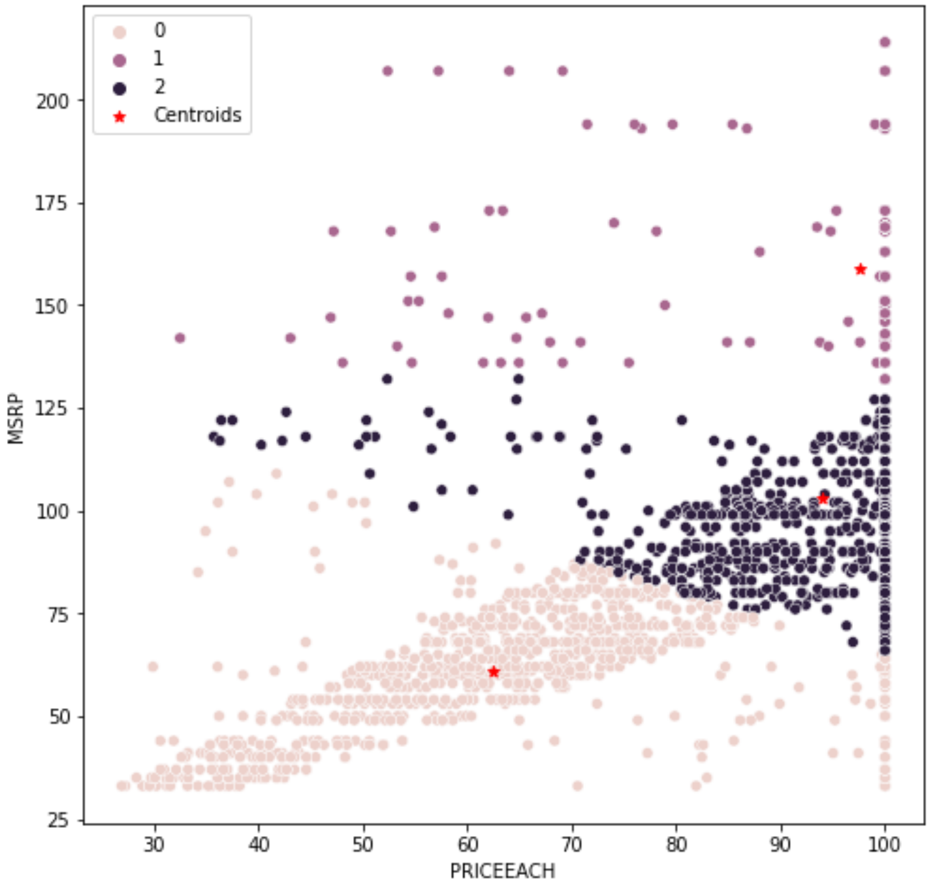


In [12]:
```
kmeans = KMeans(n_clusters = 3, random_state = 42)
y_kmeans = kmeans.fit_predict(df)
y_kmeans
```

Out[12]:
```
array([2, 2, 2, ..., 0, 0, 0], dtype=int32)
```

In [13]:
```
plt.figure(figsize=(8,8))
sns.scatterplot(x=df['PRICEEACH'], y=df['MSRP'], hue=y_kmeans)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c = 'red', label = 'Centroids
plt.legend()
```

Out[13]:
```
<matplotlib.legend.Legend at 0x7f9a64686b60>
```



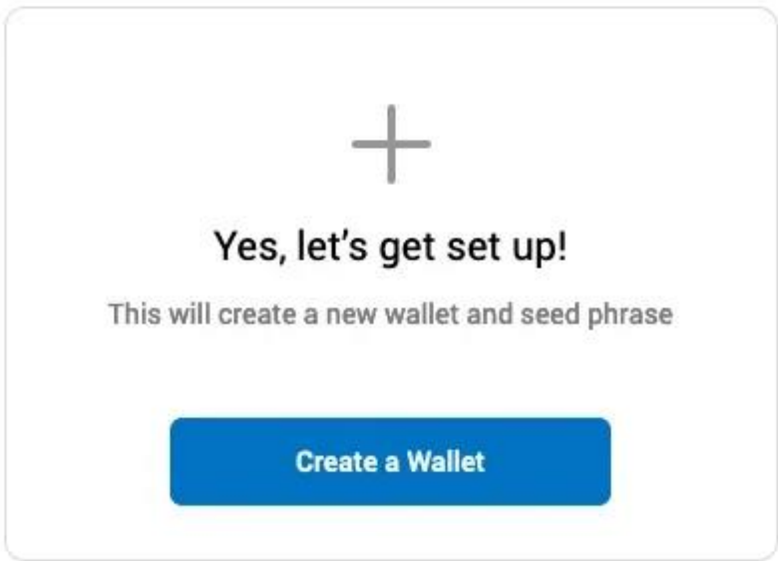
In [14]:
```
kmeans.cluster_centers_
```

Out[14]:
```
array([[ 62.49548902,  60.71556886],
       [ 97.59890263, 158.7202473 ],
       [ 94.03841567, 102.88841567]])
```

BT:

1) Installation of MetaMask and study spending Ether per transaction

Code:



Help Us Improve MetaMask

MetaMask would like to gather usage data to better understand how our users interact with the extension. This data will be used to continually improve the usability and user experience of our product and the Ethereum ecosystem.

MetaMask will..

- ✓ Always allow you to opt-out via Settings
- ✓ Send anonymized click & pageview events
- ✓ Maintain a public aggregate dashboard to educate the community

- ✗ **Never** collect keys, addresses, transactions, balances, hashes, or any personal information
- ✗ **Never** collect your full IP address
- ✗ **Never** sell data for profit. Ever!

No Thanks | I agree

This data is aggregated and is therefore anonymous for the purposes of General Data Protection Regulation (EU) 2016/679. For more information in relation to our privacy practices, please see our Privacy Policy here.

Create Password

New password (min 8 chars)

••••••••••••••••••

Confirm password

••••••••••••••••••

☑ I have read and agree to the Terms of Use

Create

Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

CLICK HERE TO REVEAL SECRET WORDS

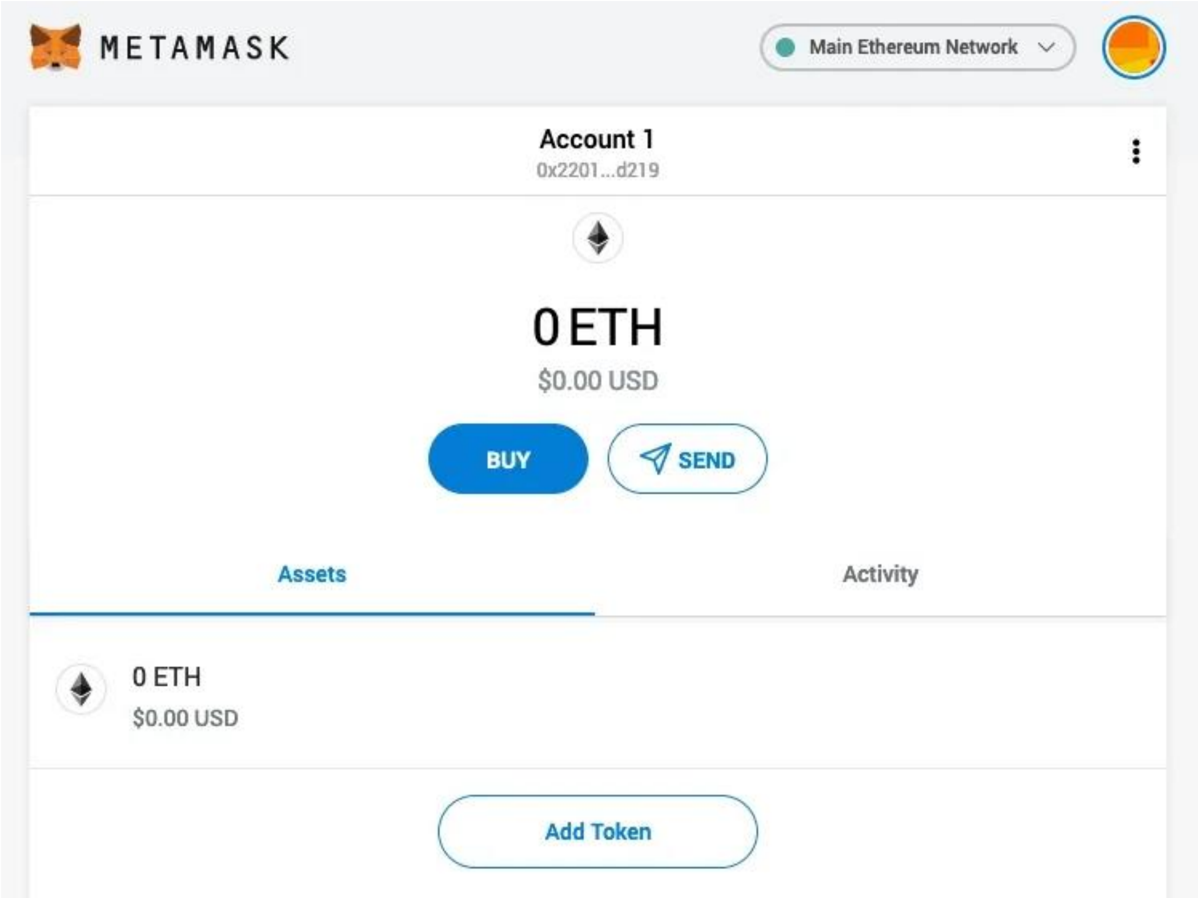
Remind me later

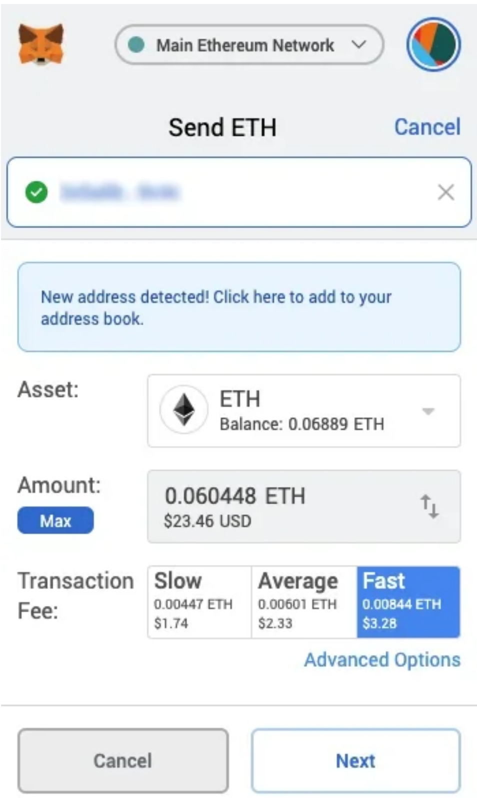
Next

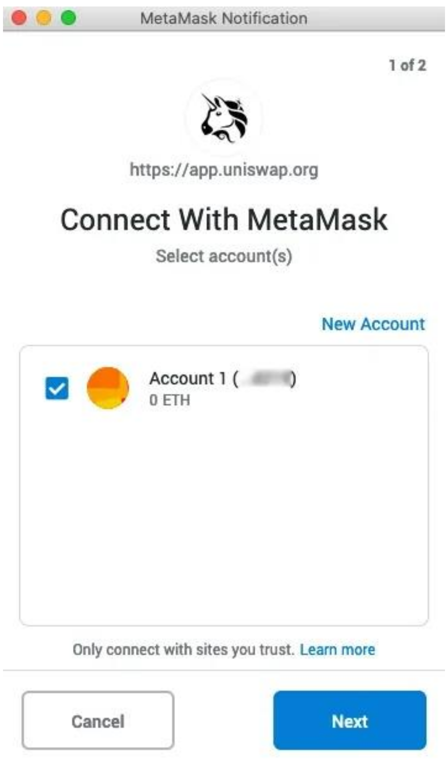
Confirm your Secret Backup Phrase

Please select each phrase in order to make sure it is correct.

always	carbon	entry	glove
invite	ladder	minor	prefer
replace	sight	six	tribe

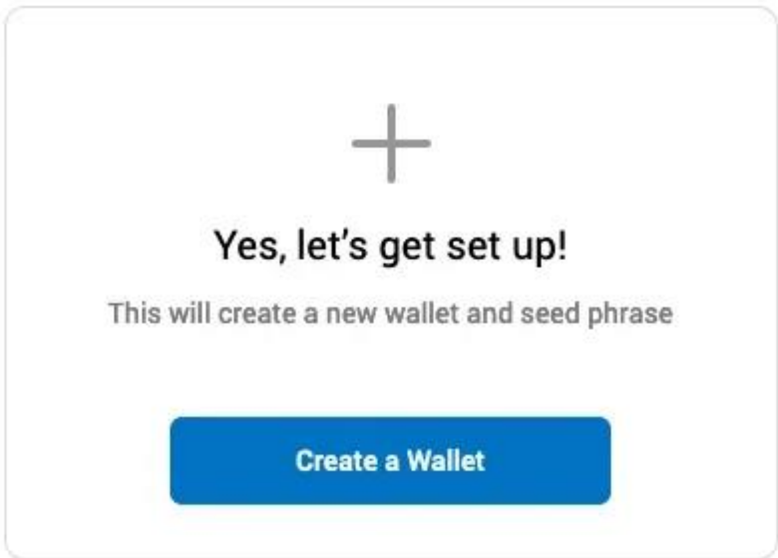

METAMASK
Main Ethereum Network
Account 1
0x2201...d219
0 ETH
$0.00 USD
BUY
SEND
Assets
Activity
0 ETH
$0.00 USD
Add Token


Main Ethereum Network
Send ETH
Cancel
New address detected! Click here to add to your address book.
Asset:
ETH
Balance: 0.06889 ETH
Amount:
Max
0.060448 ETH
$23.46 USD
Transaction Fee:
Slow
0.00447 ETH
$1.74
Average
0.00601 ETH
$2.33
Fast
0.00844 ETH
$3.28
Advanced Options
Cancel
Next


MetaMask Notification
1 of 2
https://app.uniswap.org
Connect With MetaMask
Select account(s)
New Account
Account 1 ( )
0 ETH
Only connect with sites you trust. Learn more
Cancel
Next

2) Create your own wallet using Metamask for crypto transactions

Code:



Help Us Improve MetaMask

MetaMask would like to gather usage data to better understand how our users interact with the extension. This data will be used to continually improve the usability and user experience of our product and the Ethereum ecosystem.

MetaMask will..

- ✓ Always allow you to opt-out via Settings
- ✓ Send anonymized click & pageview events
- ✓ Maintain a public aggregate dashboard to educate the community

- ✗ **Never** collect keys, addresses, transactions, balances, hashes, or any personal information
- ✗ **Never** collect your full IP address
- ✗ **Never** sell data for profit. Ever!

No Thanks | I agree

This data is aggregated and is therefore anonymous for the purposes of General Data Protection Regulation (EU) 2016/679. For more information in relation to our privacy practices, please see our Privacy Policy here.

Create Password

New password (min 8 chars)

••••••••••••••••••••

Confirm password

••••••••••••••••••••

☑ I have read and agree to the Terms of Use

Create

Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

CLICK HERE TO REVEAL SECRET WORDS

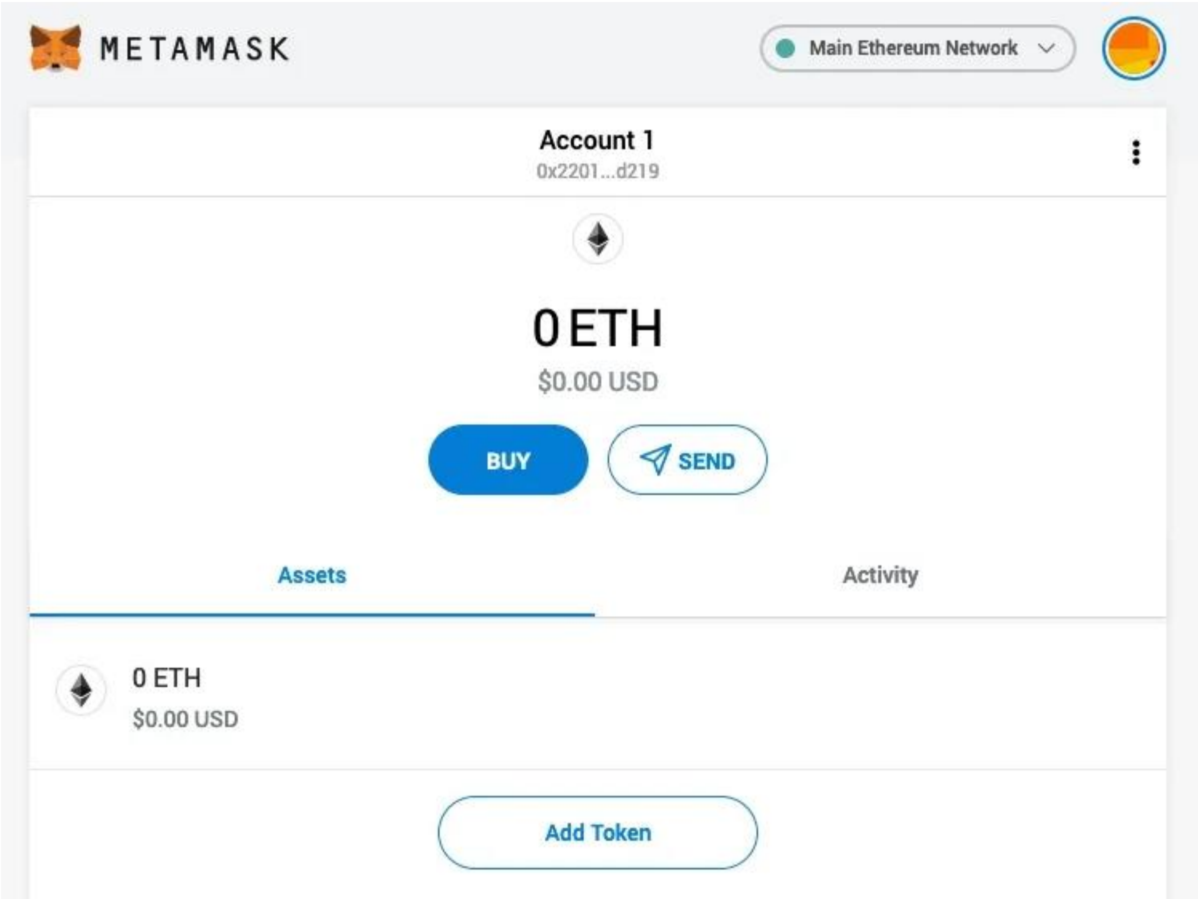
Remind me later

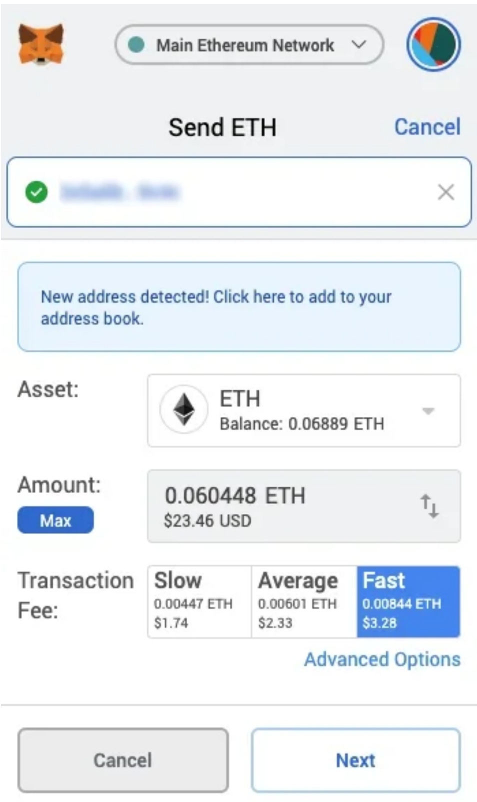
Next

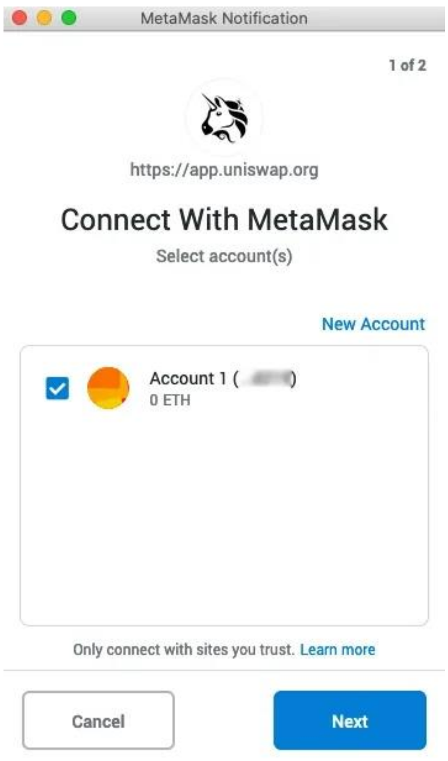
Confirm your Secret Backup Phrase

Please select each phrase in order to make sure it is correct.

always	carbon	entry	glove
invite	ladder	minor	prefer
replace	sight	six	tribe


METAMASK
Main Ethereum Network
Account 1
0x2201...d219
0 ETH
$0.00 USD
BUY
SEND
Assets
Activity
0 ETH
$0.00 USD
Add Token


Main Ethereum Network
Send ETH
Cancel
New address detected! Click here to add to your address book.
Asset:
ETH
Balance: 0.06889 ETH
Amount:
Max
0.060448 ETH
$23.46 USD
Transaction Fee:
Slow
0.00447 ETH
$1.74
Average
0.00601 ETH
$2.33
Fast
0.00844 ETH
$3.28
Advanced Options
Cancel
Next

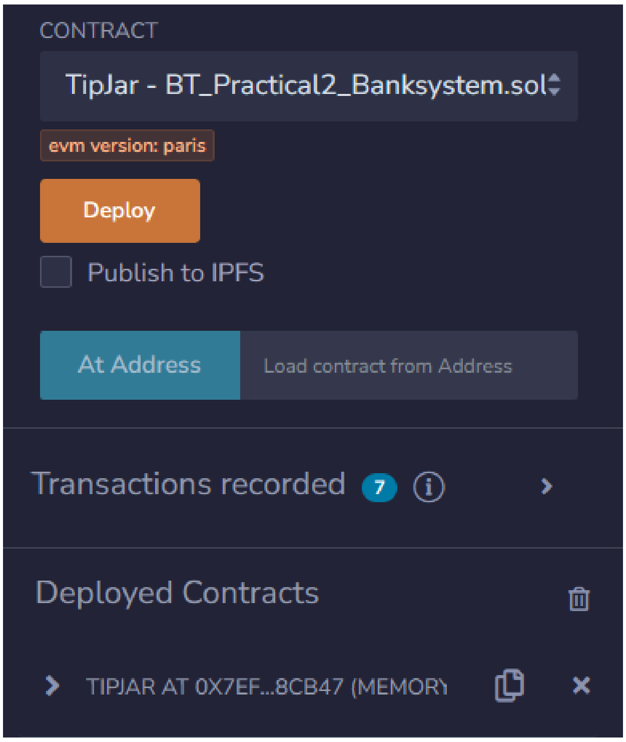

MetaMask Notification
1 of 2
https://app.uniswap.org
Connect With MetaMask
Select account(s)
New Account
Account 1 ( )
0 ETH
Only connect with sites you trust. Learn more
Cancel
Next

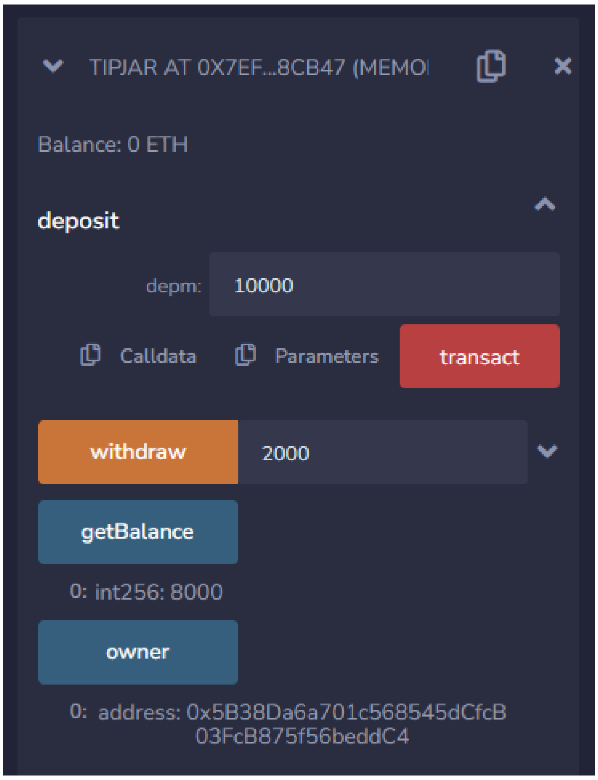
Practical 3 Smart Contact for Bank system :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract TipJar {
   int depmoney;
   int withdraw_trans;
   int balance;
   address public owner; // Current owner of the contract

   constructor() {
      owner = msg.sender;
   }

   modifier onlyOwner() {
      require(msg.sender == owner, "Only the owner can call this function");
      _;
   }

   function withdraw(int witm) public onlyOwner {
      withdraw_trans= witm;
      depmoney=depmoney-witm;
      payable(owner).transfer(address(this).balance);
   }

   function deposit(int depm) public payable {
     depmoney = depm;
```

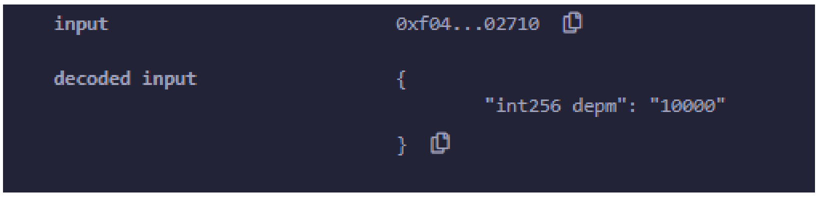
```
    // No need to specify an amount; the function should be called with the desired value.
  }

  function getBalance() public view returns (int256) {
    //balance = depmoney;
    return depmoney;
   //address(this).balance;
  }
}
```

Output Screenshots :

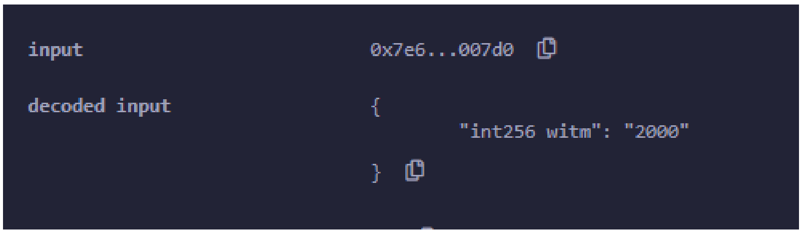




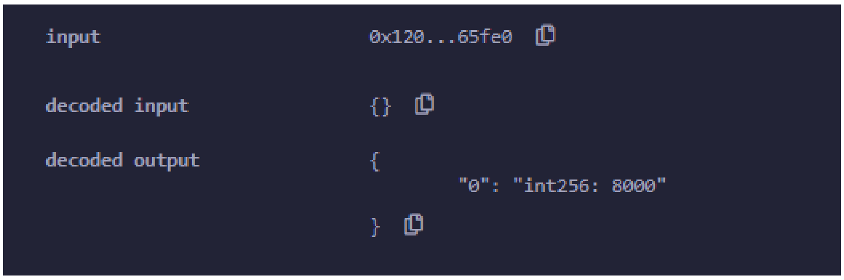
Deposit :



WithDraw :



Balance :

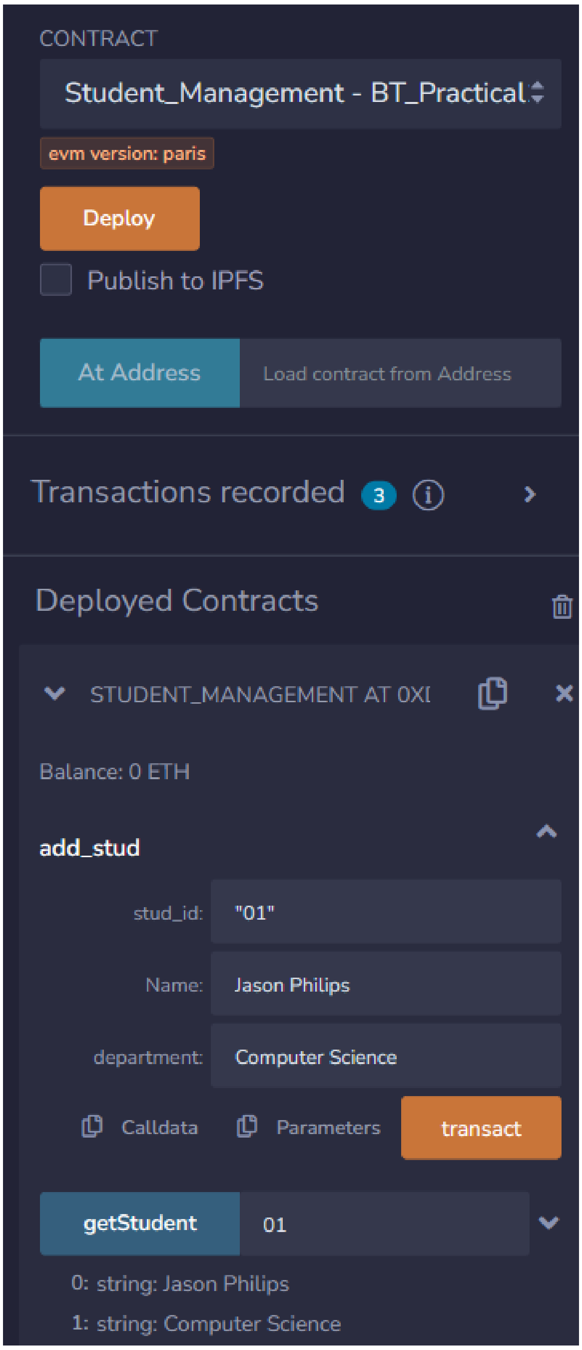


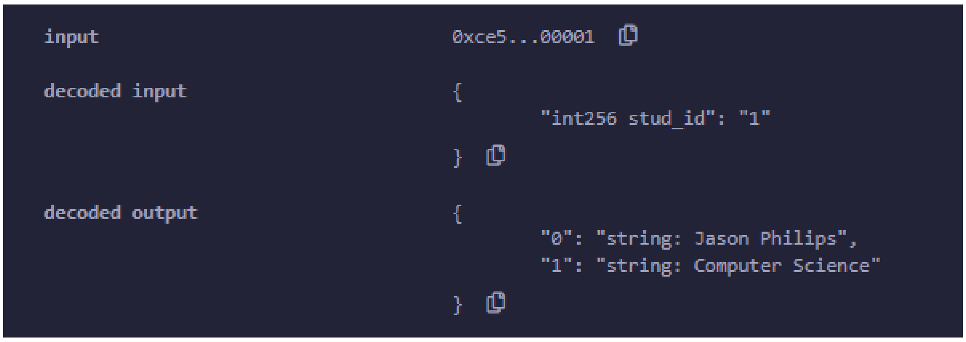
Blockchain Technology **Practical 4 : Program in solidity to create student data**

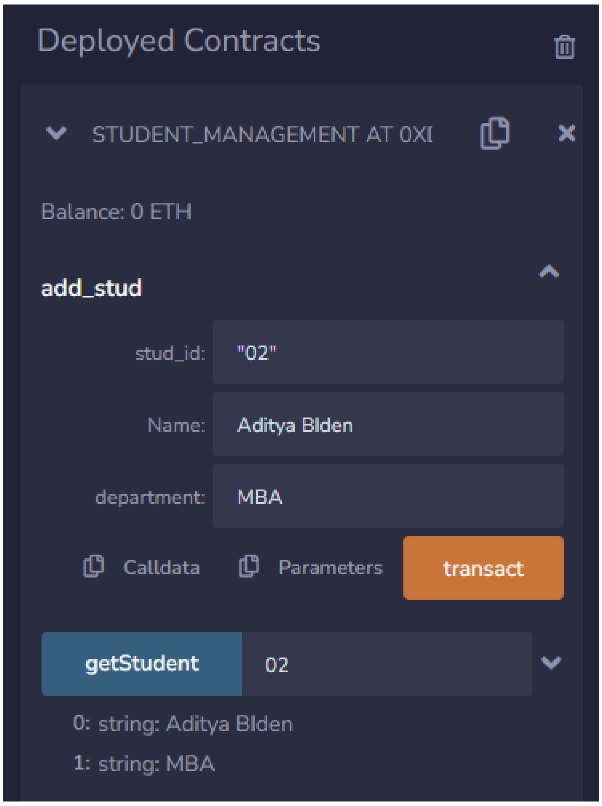
Program Code :

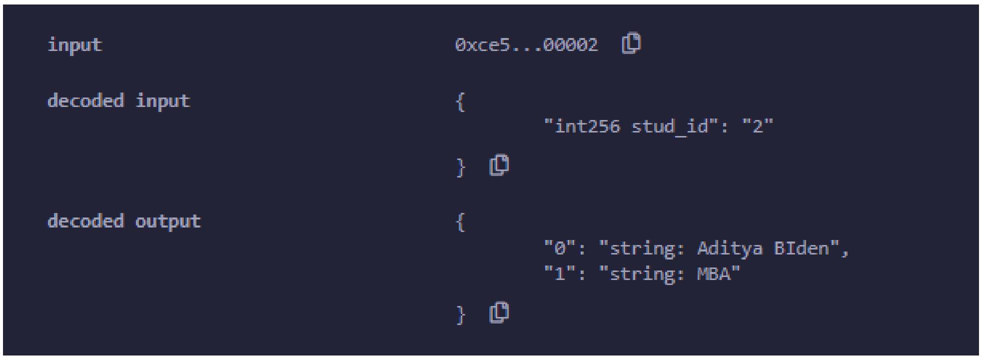
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract Student_Management{
  struct Student{
    int stud_id;
    string name;
    string department;
  }
  Student[] Students;
  function add_stud(int stud_id,string memory Name, string memory department) public{
    Student memory stud = Student(stud_id,Name,department);
    Students.push(stud);
  }

  function getStudent(int stud_id) public view returns(string memory, string memory) {

    for (uint i= 0;i<Students.length;i++){
      Student memory stud=Students[i];
      if(stud.stud_id == stud_id){
        return (stud.name,stud.department);
      }
    }
    return ("Not Found","Not Found");
  }
}
```

Console Output:


CONTRACT
Student_Management - BT_Practical
evm version: paris
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 3
Deployed Contracts
STUDENT_MANAGEMENT AT 0XI
Balance: 0 ETH
add_stud
stud_id: "01"
Name: Jason Philips
department: Computer Science
Calldata
Parameters
transact
getStudent 01
0: string: Jason Philips
1: string: Computer Science


input 0xce5...00001
decoded input {
"int256 stud_id": "1"
}
decoded output {
"0": "string: Jason Philips",
"1": "string: Computer Science"
}


Deployed Contracts
STUDENT_MANAGEMENT AT 0XI
Balance: 0 ETH
add_stud
stud_id: "02"
Name: Aditya BIden
department: MBA
Calldata
Parameters
transact
getStudent
02
0: string: Aditya BIden
1: string: MBA


input
0xce5...00002
decoded input
{
"int256 stud_id": "2"
}
decoded output
{
"0": "string: Aditya BIden",
"1": "string: MBA"
}