The approach here:

1. every row in a dataframe is converted to tupple
2. Every record is then inserted to the table using pyodbc

```
params = 'DRIVER='+driver + ';SERVER='+server +
';PORT=1433;DATABASE=' + database + ';UID=' + username + ';PWD=' +
password

#df_op is the dataframe that needs to be written to database and test
is the table name in database and col_name1, col_name2,... are the
respective column names

cnxn = pyodbc.connect(params)
cursor = cnxn.cursor()
for row_count in range(0, df_op.shape[0]):
      chunk = df_op.iloc[row_count:row_count + 1,:].values.tolist()
      tuple_of_tuples = tuple(tuple(x) for x in chunk)
      cursor.executemany("insert into test" + " ([col_name1],
col_name2],[col_name3],[col_name4],[col_name5],[col_name6],
[col_name7],[col_name8],[col_name9],[col_name10]) values
(?,?,?,?,?,?,?,?,?,?)",tuple_of_tuples)
```

Please find the respective rowcounts of a data frame and time taken to write to database using this method,

rows_count=['50','1000','5000', '0.01M','0.05M','0.1M','0.2M','0.3M']
time(sec)= [0.005, 0.098, 0.440, 0.903, 4.290, 8.802, 17.776, 26.982]

Method2

Now lets add cursor.fast_executemany = True to the function already used in method1. difference between method1 and method2 is highlighted

```
#df_op is the dataframe that needs to be written to database and test
is the table name in database and col_name1, col_name2,... are the
respective column names


cnxn = pyodbc.connect(params)
cursor = cnxn.cursor()
cursor.fast_executemany = True
for row_count in range(0, df_op.shape[0]):
   chunk = df_op.iloc[row_count:row_count + 1,:].values.tolist()
   tuple_of_tuples = tuple(tuple(x) for x in chunk)
   cursor.executemany("insert into test" + " ([col_name1],
col_name2],[col_name3],[col_name4],[col_name5],[col_name6],
[col_name7],[col_name8],[col_name9],[col_name10]) values
(?,?,?,?,?,?,?,?,?,?)",tuple_of_tuples)
```

Please find the number of rows in a data frame and respective time taken to write to database using this method,
rows_count =['50','1000','5000', '0.01M','0.05M','0.1M','0.2M','0.3M']
time(sec) = [0.009, 0.179, 0.574, 1.35, 6.718, 14.949, 28.422, 42.230]

Method3:

writes dataframe df to sql using pandas ‘to_sql’ function, sql alchemy and python

```
db_params = urllib.parse.quote_plus(params)
engine = sqlalchemy.create_engine("mssql+pyodbc:///?odbc_connect=
{}".format(db_params))

#df is the dataframe; test is table name in which this dataframe is
#inserted
df.to_sql(test,engine,index=False,if_exists="append",schema="dbo")
```

Please find the number of rows in a data frame and respective time taken to write to database using this method,
rows_count=[‘50’,’1000',’5000', ‘0.01M’,’0.05M’,’0.1M’,’0.2M’,’0.3M’]
time(sec)= [0.0230, 0.081, 0.289, 0.589, 3.105, 5.74, 11.769, 20.759]

Method4:

Now lets set ***cursor.fast_executemany = True*** using events and write to database using to_sql function.(difference between method3 and method4 is highlighted)

```
from sqlalchemy import event


@event.listens_for(engine, "before_cursor_execute")
def receive_before_cursor_execute(
       conn, cursor, statement, params, context, executemany
        ):
            if executemany:
                cursor.fast_executemany = True

df.to_sql(tbl, engine, index=False, if_exists="append", schema="dbo")
```

Please find the number of rows in a data frame and respective time taken to write to database using this method,
rows_count =[‘50’,’1000',’5000', ‘0.01M’,’0.05M’,’0.1M’,’0.2M’,’0.3M’]
time(sec)= [0.017, 0.015, 0.031, 0.063, 0.146, 0.344, 0.611, 0.833]

Now, lets compare the time taken by different methods to write to database for inserting dataframes with different sizes (ranging from 50 to 0.3 million records). ‘rows count’ represents number of rows written to dataframe, ‘time’ represents time taken by different methods to insert the respective nunber of rows to database