1. Problem statement:

- How can we use Machine Learning constructs, to implement an intelligent system, that allow the user to see a predicted forecast of resources. Input to the system is resource log files. The system is expected to perform required data cleaning and transformation. Then the system is expected to perform required analysis on the processed data and generate results in terms of future predictions. The output of the system should be a predicted resources log.

2. Converting Business problem into Machine learning problem:

- **Business Problem :** For any cloud service provider business, forecasting various resource attributes like ‘memory consumption’, ‘cpu consumption’ at a given time, is a useful tool to conduct strategic and optimization planning.

- **ML Problem :** Given resource utilization logs (.txt files) per instance, we want to predict future values for each instance for each attribute. Resource attributes are : ['Memory_Used’, ‘CPU_Used’, ‘Network_bandwidth_utilization’, ‘Storage_space_utilization’]

3. Brief description: Understanding the structure of a dataset.

a. As part of this exercise, raw data is provided i.e., data is not in the form of ‘.csv’ or database files.
b. After extracting the ‘.zip’ file we may find one folder - ‘group82_resource_utilization’ which contains 3570 subfolders in it.
c. Each subfolder has one resource utilization log (mem.log) file belonging to each instance. Name of the folder is as follow,
    i. Folder : ‘group_1_1b6ffb4a-b7bc-48d0-ab60-b43f64b7c6f4’.
    ii. ‘group_1’ is group_name and ‘1b6ffb4a-b7bc-48d0-ab60-b43f64b7c6f4’ is instance name
d. The structure of mem.log file is as follow,
    i. "{timestamp}”:”{Memory Allocated}:{Memory Used}:{CPU Allocated}:{CPU Used}:{Network bandwidth utilization}:{Storage space utilization}".
    ii. Each file can have a different number of records in it.

4. Brief description: Understanding the problem

End objective - As part of this assignment, we are supposed to build a model which can predict resource utilization per instance or per group. When a ML model predicts real values as output we call it a ‘Regression’ problem.

- This regression problem is related to ‘Time’ feature i.e., 2nd value of any attribute is dependent on 1st value of corresponding attribute. Each data point is provided with a timestamp.
- Such a problem is known as a time-series problem.

Data pipeline - Given data is in raw format, Hence we need to design a data pipeline which can input data from folders, subfolders, .txt files and output it into structured 'DataFrame' or 'ndarrays'.

- We need to analyze and clean up data very carefully in order to construct a successful ML model, usually raw data may contain lots of junk and unnecessary information.

5. Proposed solution -

Feature Transformation : In the entire machine learning project cycle, feature extraction and transformation is the most critical and important task.

- After spending good time on Exploratory data analysis (EDA) I found - **`[Memory_Used, CPU_Used, Network_bandwidth_utilization, Storage_space_utilization]`** are the most useful features for the task.
- Most of the instances have 'Network_bandwith_utilization' and 'Storage_space_utilization' constants. For regression tasks, we need to have variance in target_feature. Hence, consider 'Memory_Used' and 'CPU_Used' as target features and neglect 'Network_bandwith_utilization' and 'Storage_space_utilization' target features.
- For any time series forecasting problem, very basic yet powerful features are **'Lag_features'** i.e., in order to predict value for a 6th timestep we need a sequence of previous timestep values with some window. We will use 'Lag_features' for both problems - 'Instance based' and 'Group based' models.
- I did experiments with window value and finally chose 'window = 15', i,e., consider values of the last 15 timesteps to predict 16th position.

Performance Metric : For a problem we are dealing with, **'root_mean_squared_error'** and **'mean_absolute_error'** are very suitable metrics - since it's a regression problem.

Loss function :

1. **Mean_squared_error -**

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Advantage: The MSE is great for ensuring that our trained model has no outlier predictions with huge errors, since the MSE puts larger weight on these errors due to the squaring part of the function.
- Disadvantage: If our model makes a single very bad prediction, the squaring part of the function magnifies the error.

2. Median_absolute_error -

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

- **Advantage** - unlike the MSE, we won't be putting too much weight on our outliers and our loss function provides a generic and even measure of how well our model is performing.
- **Disadvantage**: If we care about the outlier predictions of our model, then the MAE won't be as effective. The large errors coming from the outliers end up being weighted the exact same as lower errors.

- **3. Huber_loss -**

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

- This equation says : for loss values less than delta, use the MSE; for loss values greater than delta, use the MAE. This effectively combines the best of both worlds from the two loss functions. Using the MAE for larger loss values mitigates the weight that we put on outliers so that we still get a well-rounded model. At the same time we use the MSE for the smaller loss values to maintain a quadratic function near the centre.

Modelling :

- For any time-series data LSTM (Long_short_term_memory cells) and GRU (Gated_recurrent_units) are state of the art deep learning models.

Ideal case -

- Ideally, we should build a baseline model at first stage i.e., simple_moving_avg, exponential_weighted_avg etc. and measure performance metric on it.
- At the second stage measure metrics on basic and complex 'ML' models like 'Linear_regression', 'Random_forest', 'GBDT' etc.
- Third, train state_of_the_art (SOTA) models like RNN, LSTM, GRU etc. again measure performance of the model.

- Now, compare the model's performance using metrics, also consider the computational resources we need if we deployed the model. In simple words, choosing a final model is nothing but a trade off between ‘performance_metric’, ‘time complexity’ and ‘space_complexity’ for final output.
- Finally tweak the performance of a final model by doing some research and experiments.

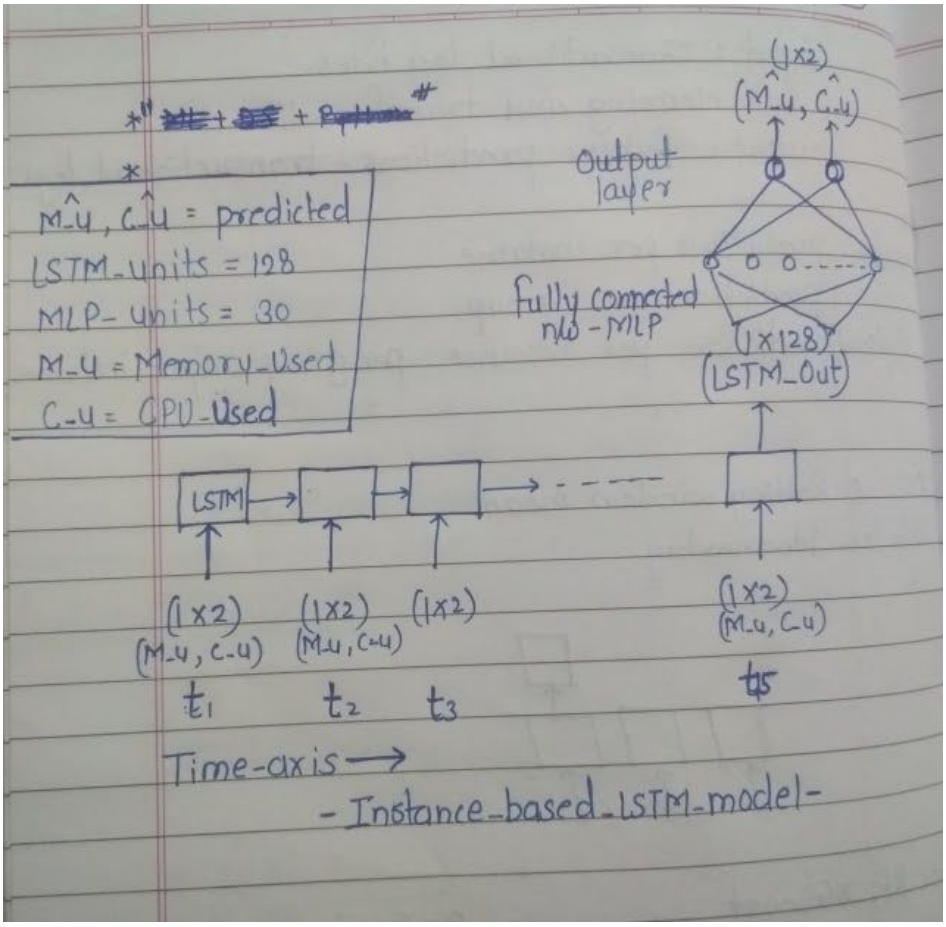
Assignment case -

- To keep things simple for this exercise, Also to showcase my deep learning (SOTA) algorithms (Tensorflow and Keras) skills, I chose to go with LSTM. Since, LSTM is one of the most powerful algorithms when it comes to time-series data. With the help of MLP (Multi_layer_perceptron), LSTM can handle multi-label regression problems very easily i.e, at a single time model can predict multiple target features - ‘cpu_used’ and ‘memory_used’.

Instance based model -

- In this experiment of an instance based model, I took data from an instance with 49K data points in total. 34.5K (70%) goes into the train_set and 14.5K (30%) data points goes into the validation set.
- For time-series data we cannot employ random train_test_splitting techniques. Here I used time-based-splitting for the effectiveness.
- And built a model for two target features (‘cpu_used’ and ‘memory_used’) at single time i.e., multi-label regression case.

- Model architecture -



Group based model -

- For the group based model, I took data from an instance with 640K data points in total. 448K (70%) goes into the train_set and 192K (30%) data points goes into the validation set.
- Group data is large data as compared to a single instance data, hence we need to treat it slightly differently. Usually large dataset may contain outlier data points. Generally, outliers impact heavily on the performance of the models, to remove outliers I used simple statistics like PDF, CDF and percentiles.
- Since data is coming from multiple instances, we need to create some features to provide this information to the model. Most popular method is One_hot_encoding. I created OHE features on ‘instance_name’ and provided directly to the dense (MLP) layer. The architecture of the ‘Instance_based_model’ and ‘Group_based_model’ is exactly the same, except OHE features.
- Also, I trained 2 different models for ‘Memory_Used’ and ‘CPU_Used’ features, because 'Memory_Used' target feature has real (integer) values greater than 512 i.e., very large values and 'CPU_Used' target feature

has real (float) values ranging between 0 to some small values (e.g. 20) i.e., it has very small float values as compared to the 'Memory_used' feature.

6. Test results :

Instance_based_model : [Memory_used, CPU_used]

```
# 8. Predictions
idx = 145
x_test = [dense_x_cv[idx].reshape(1, -1), lstm_x_cv[idx].reshape(1, 15, 2)]
y_pred = model.predict(x_test)
y_true = y_cv[idx]

print('y_pred :', y_pred, ' | y_true :', y_true)
```

```
y_pred : [[3.9101667e+03 3.8252997e-01]]  | y_true : [3.979e+03 2.200e-01]
```

Group_based_model : [Memory_used]

```
# 8. Predictions
idx = 104
x_test = [grp_dense_x_cv[idx].reshape(1, -1), np.expand_dims(\
          grp_lstm_x_cv[:, :, 1],axis = -1)[idx].reshape(1, 15, 1)]
y_pred = cpu_model.predict(x_test)
y_true = grp_y_cv[idx][1]
print('predicted :', y_pred, ' | y_true :', y_true)
```

```
predicted : [[0.01807757]]  | y_true : 0.06
```

Group_based_model : [CPU_used]

```
# 8. Predictions
idx = 145
x_test = [grp_dense_x_cv[idx].reshape(1, -1), np.expand_dims(\
          grp_lstm_x_cv[:, :, 0], axis = -1)[idx].reshape(1, 15, 1)]
y_pred = memory_model.predict(x_test)
y_true = grp_y_cv[idx][0]
print('predicted :', y_pred, ' | y_true :', y_true)
```

```
predicted : [[14626.854]]  | y_true : 16382.0
```

Final performance table :

Models	train_rmse	val_rmse	train_mse	val_mse	train_mae	val_mae
Instance_based_model	131.7863	99.2126	-	-	-	-
Group_based_model (CPU)	0.243	0.2005	0.059	0.0402	0.1125	0.0939
Group_based_model (mem)	-	-	-	-	3699.032	1861.3419

7. Future scope:

a. Better we train different models on each target feature.
b. We need to employ baseline models for better comparison i.e., Train simple ML models like linear regression and some complex models like Random forest, GBDT etc. to compare performance of models with Deep learning models.
c. We need to experiment with more time series feature transformation techniques like, i. Fourrier transform
   ii. Simple moving average
   iii. Exponential weighted moving avg
   iv. Log transforms (usually work best on features with large values e.g., memory_used
   v. Box-cox transform (to convert features into normal/gaussian distribution)
d. To build the robust model, better we utilize all the provided data and train a LSTM model on top of it. Also we need to perform good EDA to remove outlier data points.
e. Experiment with various window sizes for lag features to improve the results.