Experiment tracking

The usual process for building an EndToEnd machine learning project involves collecting and processing raw data, analyzing it features at steps, training different algorithms, evaluating them, and deploying the best model on some platform for user access. It seems fairly straightforward, right? But in reality it is not. There are several complexities that arise along the way. Due to the circular nature of this process, its more about experimenting and trying out different things that may work

- ML is not just code. It is code plus data that we need to keep a track of. Data can be sourced from multiple storage units
- use different models and model hyperparameters
- run the same code in a different environment
- Model governance is another important aspect, where everything starting from experimentation to deployment is tracked for auditing purposes, where models are tested for speed, accuracy, drift while in production to avoid inaccuracy.

You can think of experiments as the process of building an ML model. When we say experiment run, we mean each trial in an ML experiment. So the ML experiment is actually the whole process that a data scientist may start playing with some data, models and hyperparameters. Each of these trials is an experiment run.

Experiment tracking is the process of keeping track of all the relevant information from ML experiments.

- **Organize** all the necessary components of a specific experiment. It's important to have everything in one place and know where it is so you can use them later.
- **Reproduce** past results (easily) using saved experiments.
- **Log** iterative improvements across time, data, ideas, teams, etc.

If you are working in a finance company and are tasked with creating a ml model that based on certain conditions classify if the applicant should be given a loan or not

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
train_df = pd.read_csv('data.csv')
train_df.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
0	LP002529	Male	Yes	2	Graduate	No	6700	1750.0	230.0	300.0	1
1	LP001385	Male	No	0	Graduate	No	5316	0.0	136.0	360.0	1
2	LP001926	Male	Yes	0	Graduate	No	3704	2000.0	120.0	360.0	1
3	LP001144	Male	Yes	0	Graduate	No	5821	0.0	144.0	360.0	1
4	LP002562	Male	Yes	1	Not Graduate	No	5333	1131.0	186.0	360.0	Na

In [3]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            430 non-null    object
 1   Gender             420 non-null    object
 2   Married            427 non-null    object
 3   Dependents         416 non-null    object
 4   Education          430 non-null    object
 5   Self_Employed      410 non-null    object
 6   ApplicantIncome    430 non-null    int64
 7   CoapplicantIncome  430 non-null    float64
 8   LoanAmount         414 non-null    float64
 9   Loan_Amount_Term   422 non-null    float64
 10  Credit_History     394 non-null    float64
 11  Property_Area      430 non-null    object
 12  Loan_Status        430 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 43.8+ KB
```

Binary Encoding of Categorical Variables

In [4]:

```
train_df['Gender']= train_df['Gender'].map({'Male':0, 'Female':1})
train_df['Married']= train_df['Married'].map({'No':0, 'Yes':1})
train_df['Loan_Status']= train_df['Loan_Status'].map({'N':0, 'Y':1})
```

In [5]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            430 non-null    object 
 1   Gender             420 non-null    float64
 2   Married            427 non-null    float64
 3   Dependents         416 non-null    object 
 4   Education          430 non-null    object 
 5   Self_Employed      410 non-null    object 
 6   ApplicantIncome    430 non-null    int64  
 7   CoapplicantIncome  430 non-null    float64
 8   LoanAmount         414 non-null    float64
 9   Loan_Amount_Term   422 non-null    float64
 10  Credit_History     394 non-null    float64
 11  Property_Area      430 non-null    object 
 12  Loan_Status        430 non-null    int64  
dtypes: float64(6), int64(2), object(5)
memory usage: 43.8+ KB
```

Checking for Missing Values

In [6]:

```
train_df.isnull().sum()
```

Out[6]:

```
Loan_ID               0
Gender               10
Married               3
Dependents           14
Education             0
Self_Employed        20
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           16
Loan_Amount_Term      8
Credit_History       36
Property_Area         0
Loan_Status           0
dtype: int64
```

In [7]:

```
## dropping all the missing values
train_df = train_df.dropna()
train_df.isnull().sum()
```

Out[7]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Segregating the target variable from the features

In [8]:

```
X = train_df[['Gender', 'Married', 'ApplicantIncome', 'LoanAmount', 'Credit_History']]
y = train_df.Loan_Status
X.shape, y.shape
```

Out[8]:

```
((335, 5), (335,))
```

In [ ]:

```

```

Splitting the data

In [9]:

```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=5)
```

Model Training

In [10]:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(max_depth=4, random_state=5)
model.fit(X_train, y_train)
```

Out[10]:

```
RandomForestClassifier(max_depth=4, random_state=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Cross Validation

In [11]:

```
from sklearn.metrics import accuracy_score

pred_val = model.predict(X_val)
accuracy_score(y_val, pred_val)
```

Out[11]:

```
0.7313432835820896
```

In [12]:

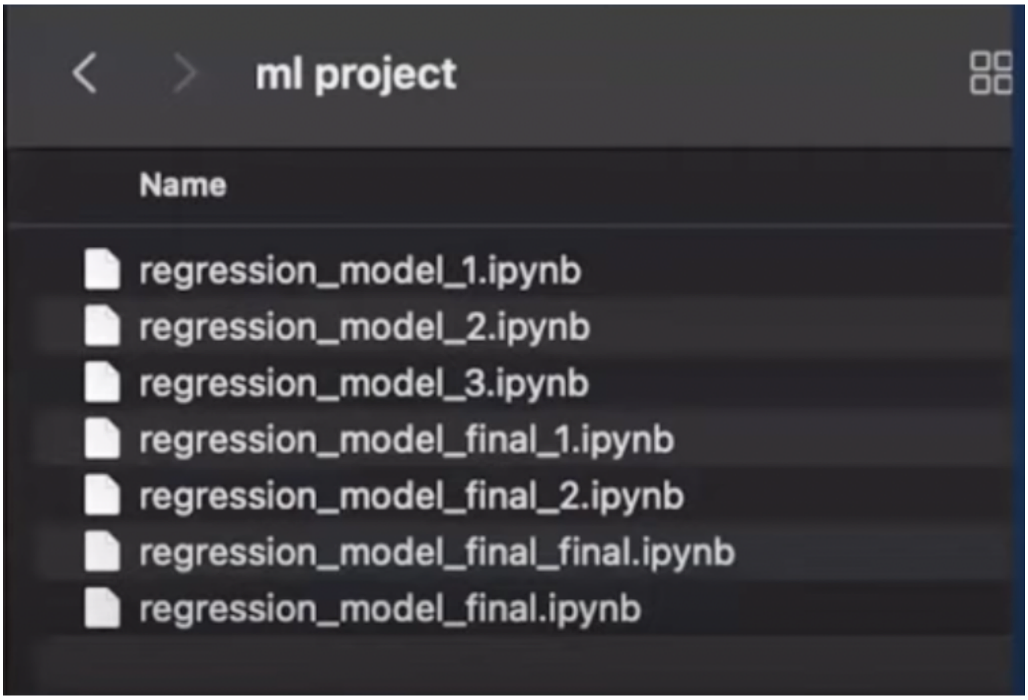
```
pred_train = model.predict(X_train)
accuracy_score(y_train, pred_train)
```

Out[12]:

```
0.8134328358208955
```

We have created a model successfully, but now we have new set of data how do you proceed on working with it

- we can change the data and run the code again
  - but we will loose the output and results from the old data
- we can create new cells below these to create a new model with the new data
  - but then when we have a lot of experiments in one file it will be really difficult finding the one we want to look at
- We can create new files for each experiment
  - but for actually comparing the rsults and outputs you'll still have to ope each file and look into it closely



these are not the best ways of keeping track of the work and experiments that you perform, we need to create something that easy to manage, clearly shows the results and metrics, logs the changes and hyperparameters for us

ML Flow

https://mlflow.org/ (https://mlflow.org/)

MLflow is an open-source platform to manage Machine Learning Lifecycle. In layman's terms, it can track and store data, parameters, and metrics to be retrieved later or displayed nicely on a web interface.Furthermore, MLflow is a framework-agnostic tool, so any ML / DL framework can quickly adapt to the ecosystem that MLflow proposes.

MLflow emerges as a platform that offers tools for tracking metrics, artifacts, and metadata.

ML flow Tracking

MLflow Tracking is an API-based tool for logging metrics, parameters, model versions, code versions, and files. MLflow Tracking is integrated with a UI for visualizing and managing artifacts, models, files, etc.

Each MLflow Tracking session is organized and managed under the **concept of runs.**

- A run refers to the execution of code where the artifact log is performed explicitly.

By default, the runs are stored in the directory where the code session is executed. However, MLflow also allows storing artifacts on a local or remote server, for better collaboration. we'll st

getting started

In [14]:

```
!pip3 install mlflow
```

```
Requirement already satisfied: databricks-cli>=0.8.7 in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/sit
e-packages (from mlflow) (0.17.0)
Requirement already satisfied: entrypoints in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-packages
(from mlflow) (0.3)
Requirement already satisfied: alembic in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-packages (fr
om mlflow) (1.8.0)
Requirement already satisfied: docker>=4.0.0 in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-packag
es (from mlflow) (5.0.3)
Requirement already satisfied: pandas in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-packages (fro
m mlflow) (1.4.1)
Requirement already satisfied: requests>=2.17.3 in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-pac
kages (from mlflow) (2.27.1)
Requirement already satisfied: sqlalchemy in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-packages
(from mlflow) (1.4.39)
Requirement already satisfied: pyjwt>=1.7.0 in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-package
s (from databricks-cli>=0.8.7->mlflow) (2.4.0)
Requirement already satisfied: oauthlib>=3.1.0 in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-pack
ages (from databricks-cli>=0.8.7->mlflow) (3.2.0)
Requirement already satisfied: six>=1.10.0 in /Users/harshit/miniconda3/envs/dsml_env/lib/python3.9/site-packages
```

In [15]:

```
import mlflow
import os
```

we'll start by setting up our experiment name under which we wanna perform all our work

- An MLflow experiment is the primary unit of organization and access control for MLflow runs; all MLflow runs belong to an experiment. Experiment:{run,run.....run}
- Experiments let you visualize, search for, and compare runs, as well as download run artifacts and metadata for analysis in other tools.
- An MLflow run corresponds to a single execution of model code. Each run records the some information about that particulr trial:

In [16]:

```
mlflow.set_experiment("loan_status")
```

Out[16]:

```
<Experiment: artifact_location='file:///Users/abdulahad_scaler/jupyter/mlruns/1', experiment_id='1', lifecycle_st
age='active', name='loan_status', tags={}>
```

Next, you can start to think about what do you want to keep track in your analysis/experiment. MLflow categorizes these into:

- **Parameters** (via mlflow.log_param()). Parameters are variables that you change or tweak when tuning your model.
- **Metrics** (using mlflow.log_metric()). Metrics are values that you want to measure as a result of tweaking your parameters. Typical metrics that are tracked can be items like F1 score, RMSE, MAE etc.
- **Artifacts** (using mlflow.log_artifact()). Artifacts are any other items that you wish to store. Typical artifacts to keep track of are PNGs of graphs,plots, confusion matrix, and also pickled model files

Params are something you want to tune based on the metrics, whereas tags are some extra information that doesn't necessarily associate with the model's performance. there's no hard constraint on which to use to log which; they can be used interchangeably without error.

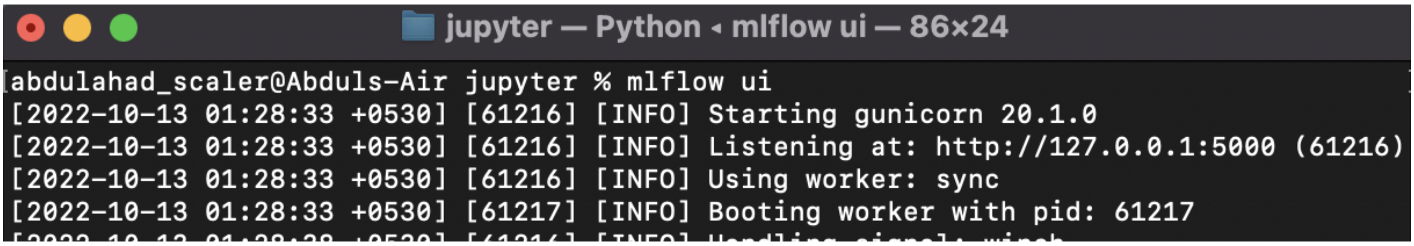
In [17]:

```
with mlflow.start_run():
    model_rf = RandomForestClassifier(max_depth=4, random_state=5)
    model_rf.fit(X_train, y_train)

    pred_val = model_rf.predict(X_val)
    val_acc=accuracy_score(y_val, pred_val)

    pred_train = model_rf.predict(X_train)
    train_acc=accuracy_score(y_train, pred_train)

    mlflow.set_tag('mlflow.runName','first_run')
    mlflow.log_param('max_depth',4)
    mlflow.log_metric('val_acc',val_acc)
    mlflow.log_metric('train_acc',train_acc)

    mlflow.sklearn.log_model(model_rf, "model")
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
/var/folders/v3/9qnnmcxd5rdbhy0swt0r_11m0000gn/T/ipykernel_43059/595822507.py in <module>
     14     mlflow.log_metric('train_acc',train_acc)
     15
---> 16     mlflow.sklearn.log_model(model_rf, "model")
     17
     18

~/miniconda3/envs/dsml_env/lib/python3.9/site-packages/mlflow/sklearn/__init__.py in log_model(sk_model, artifact
_path, conda_env, code_paths, serialization_format, registered_model_name, signature, input_example, await_regist
ration_for, pip_requirements, extra_pip_requirements)
    391         mlflow.sklearn.log_model(sk_model, "sk_models")
    392     """
--> 393     return Model.log(
    394         artifact_path=artifact_path,
    395         flavor=mlflow.sklearn,

~/miniconda3/envs/dsml_env/lib/python3.9/site-packages/mlflow/models/model.py in log(cls, artifact_path, flavor,
```

we can use this with command to start the ml flow run and whatever we do inside of that start_run indent will be tracked

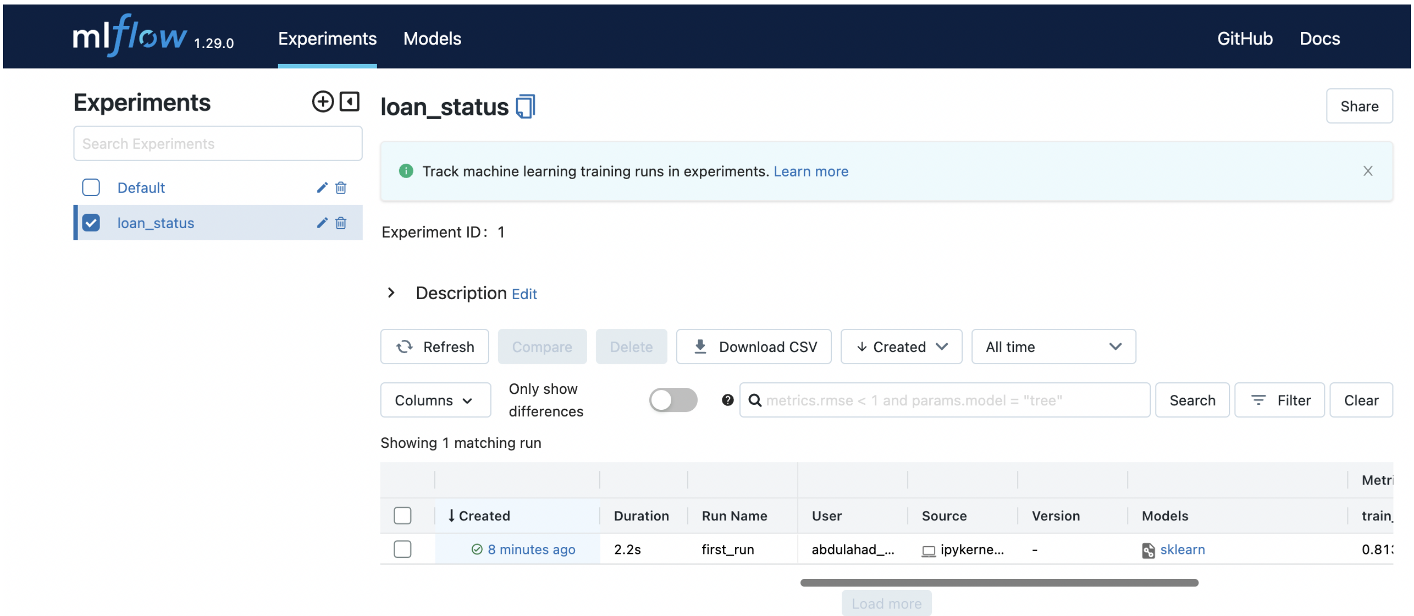
inside that we create our first model and log the different parameters and metric for that model we set the name of the run and log the max depth of the rf model and also the acc score. All of the parameters and models are stored in files in the experiment folder with each runs having seperate folders. you can open those files to see the stored data



mlflow ui

MLflow also provides the option to view all the runs and experiments on a web based ui that is really easy to use and see the logged data. Launch the MLflow tracking UI for local viewing of run results. In the folder where you have this experiments run the command **mlflow ui** this will start an ml flow ui server that is by default open at port 5000 on your localhost or 127.0.0.1 you can change the port by using -p port_num along with the command eg: **mlflow ui -p 8899**

- open the correct link or copy the provided url from the command



here is the web ui launched on a browser, as you can see we are under the loan_status experimnet name and have a run that we created with the name first_run. there are several other informations as welllike the source code that we used, the user thatcreated that run and the model that we stored loan_status › first_run

first_run

Run ID: 7d52e1ff68a34342b25ed2da8e0b69df   Date: 2022-10-13 01:20:18   Source: ipykernel_launcher.py

User: abdulahad_scaler   Duration: 2.2s   Status: FINISHED

Lifecycle Stage: active

› Description Edit

› Parameters (1)

› Metrics (2)

› Tags

⌄ Artifacts

- model
  - MLmodel
  - conda.yaml
  - model.pkl
  - python_env.yaml
  - requirements.txt

Full Path:file:///Users/abdulahad_scaler/jupyter/mlruns/1/7d52e1ff68a34342b25ed2da8e0b69df/artifacts/...

Register Model

if we click on any particular run we cansee more details about that run. we have here the all the detailsthat we logged for that particular model in a very easy to understand fashion

Parameters (1)

Name	Value
max_depth	4

Metrics (2)

Name	Value
train_acc	0.813
val_acc	0.731

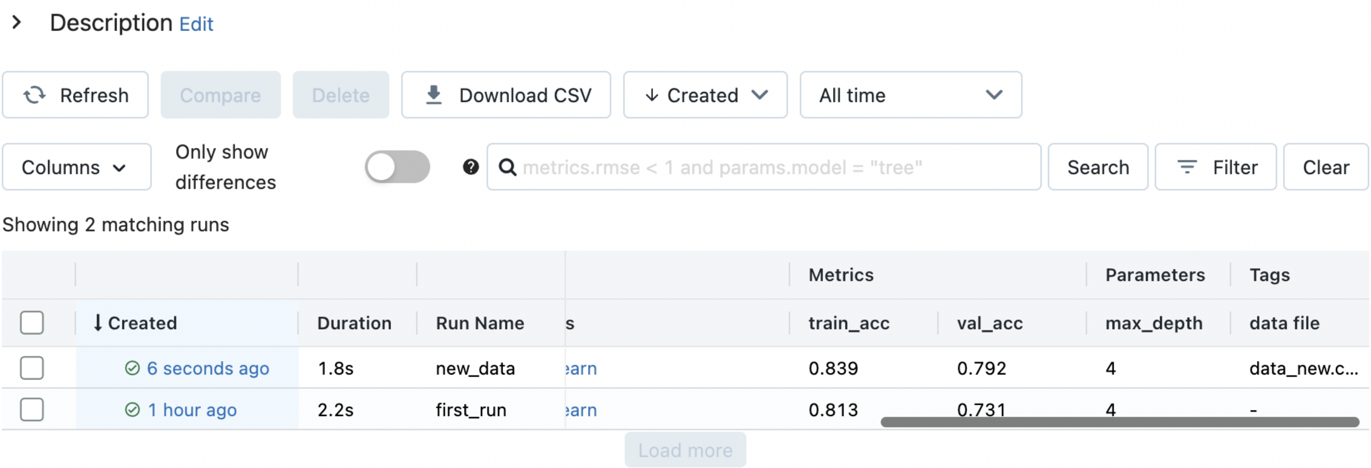
Load the new data and proceed furthur

In [53]:

```
train_df = pd.read_csv('data_new.csv')
train_df.head()

train_df['Gender']= train_df['Gender'].map({'Male':0, 'Female':1})
train_df['Married']= train_df['Married'].map({'No':0, 'Yes':1})
train_df['Loan_Status']= train_df['Loan_Status'].map({'N':0, 'Y':1})

train_df = train_df.dropna()

X = train_df[['Gender', 'Married', 'ApplicantIncome', 'LoanAmount', 'Credit_History']]
y = train_df.Loan_Status
X.shape, y.shape

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=5)
```

In [54]:

```
with mlflow.start_run():
    model_rf = RandomForestClassifier(max_depth=4, random_state=5)
    model_rf.fit(X_train, y_train)

    pred_val = model_rf.predict(X_val)
    val_acc=accuracy_score(y_val, pred_val)

    pred_train = model_rf.predict(X_train)
    train_acc=accuracy_score(y_train, pred_train)

    mlflow.set_tag('mlflow.runName','new_data')
    mlflow.log_param('max_depth',4)
    mlflow.log_metric('val_acc',val_acc)
    mlflow.log_metric('train_acc',train_acc)
    mlflow.set_tag('data file','data_new.csv')

    mlflow.sklearn.log_model(model_rf, "model")
```



if we go back to the web ui we can see that we have another run logged with the information we have we changed we added a new name and the name of the datafile

Now if we want to tune the RF model

In [61]:

```
def mlflow_runs(n_est,max_dep,i):
    with mlflow.start_run():

        model_rf = RandomForestClassifier(n_estimators=n_est, max_depth=max_dep, random_state=5)
        model_rf.fit(X_train, y_train)

        pred_val = model_rf.predict(X_val)
        val_acc=accuracy_score(y_val, pred_val)

        pred_train = model_rf.predict(X_train)
        train_acc=accuracy_score(y_train, pred_train)

        run="hyperparameter_run_"+str(i)
        mlflow.set_tag('mlflow.runName',run)
        mlflow.log_param('n_estimators',n_est)
        mlflow.log_param('max_depth',max_dep)
        mlflow.log_metric('val_acc',val_acc)
        mlflow.log_metric('train_acc',train_acc)
        mlflow.set_tag('data file','data_new.csv')

        mlflow.sklearn.log_model(model_rf, "model")
```

In [62]:

```
mlflow_runs(10,2,1)
mlflow_runs(20,2,2)
mlflow_runs(40,2,3)
mlflow_runs(10,4,4)
mlflow_runs(20,4,5)
mlflow_runs(40,4,6)
mlflow_runs(10,8,7)
mlflow_runs(20,8,8)
mlflow_runs(40,8,9)
```

Showing 13 matching runs

				Metrics		Parameters		Tags
☐	↓ Created	Duration	Run Name	train_acc	val_acc	max_depth	n_estimators	data file
☐	31 seconds ago	0.8s	hyperpara...	0.914	0.802	8	40	data_new.c...
☐	32 seconds ago	0.8s	hyperpara...	0.914	0.771	8	20	data_new.c...
☐	33 seconds ago	0.8s	hyperpara...	0.909	0.75	8	10	data_new.c...
☐	33 seconds ago	0.8s	hyperpara...	0.833	0.792	4	40	data_new.c...
☐	34 seconds ago	0.8s	hyperpara...	0.833	0.792	4	20	data_new.c...
☐	35 seconds ago	0.7s	hyperpara...	0.828	0.792	4	10	data_new.c...
☐	36 seconds ago	0.8s	hyperpara...	0.807	0.813	2	40	data_new.c...
☐	36 seconds ago	0.7s	hyperpara...	0.807	0.813	2	20	data_new.c...
☐	38 seconds ago	1.7s	hyperpara...	0.807	0.813	2	10	data_new.c...
☐	2 minutes ago	267ms	hyperpara...	-	-	-	-	-
☐	3 minutes ago	105ms	lyrical-jay-...	-	-	-	-	-
☐	38 minutes ago	1.8s	new_data	0.839	0.792	4	-	data_new.c...
☐	2 hours ago	2.2s	first_run	0.813	0.731	4	-	-

Load more

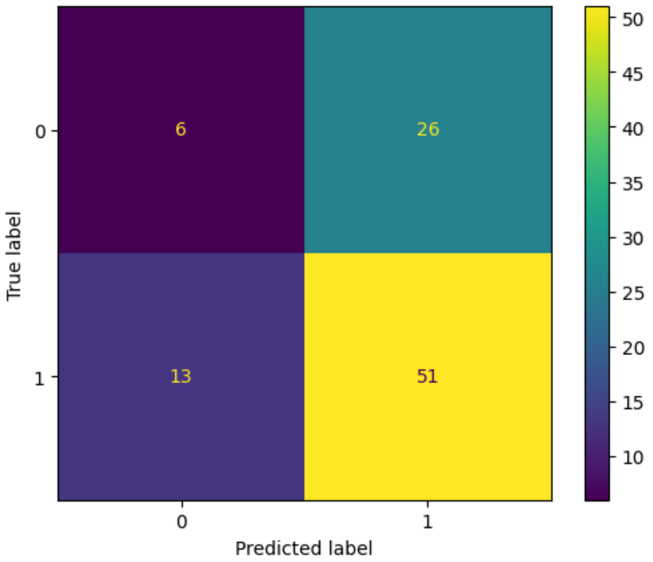
here we can see there are 9 new runs that show how our model performed

- we can see that increasing the number of tress improves the model a lot
- if we have a deep model with less number of trees it seems to overfit because the train accuracy is very high but the valaccuracy is low you can also see there are two failed runs so they have no data associated with them

now if you want to try out another model like knn for this task

In [77]:

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

with mlflow.start_run():
    knn_model= KNeighborsClassifier(n_neighbors=5)
    knn_model.fit(X_train, y_train)

    pred_val = knn_model.predict(X_val)
    val_acc=accuracy_score(y_val, pred_val)

    pred_train = knn_model.predict(X_train)
    train_acc=accuracy_score(y_train, pred_train)

    run="KNN"
    mlflow.set_tag('mlflow.runName',run)
    mlflow.log_param('neighbors',5)
    mlflow.log_metric('val_acc',val_acc)
    mlflow.log_metric('train_acc',train_acc)
    mlflow.set_tag('data file','data_new.csv')

    cm=ConfusionMatrixDisplay.from_predictions( y_val,pred_val)
    cm.figure_.savefig('confusion_mat.png')
    mlflow.log_artifact('confusion_mat.png')

    mlflow.sklearn.log_model(knn_model, "model")
```



Run ID: b97f4fabc7a649bf998547c893207d6a

Date: 2022-10-13 04:56:58

User: abdulahad_scaler

Duration: 1.7s

Lifecycle Stage: active

> Description Edit

⌄ Parameters (1)

Name	Value
neighbors	5

⌄ Metrics (2)

Name	Value
train_acc	0.766
val_acc	0.594

after all the testing and trying we can say that we will chose the random forest model with max depth =8 and number of trees=40