Import Required Library

In [2]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

In [3]:
```python
df = pd.read_csv('insurance_claims.csv')
df.head()
```

Out[3]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	polic
0	328	48	521585	2014-10-17	OH	250/500	
1	228	42	342868	2006-06-27	IN	250/500	
2	134	29	687698	2000-09-06	OH	100/300	
3	256	41	227811	1990-05-25	IL	250/500	
4	228	44	367455	2014-06-06	IL	500/1000	

Remove Unused Columns

In [4]:
```python
df.drop(columns=['policy_number', 'incident_location',
                 'auto_model', '_c39', 'policy_bind_date',
                 'incident_date', 'insured_zip', 'insured_hobbies',
                 'auto_make', 'auto_year', 'policy_annual_premium',
                 'umbrella_limit'], inplace=True)
df.head()
```

Out[4]:

	months_as_customer	age	policy_state	policy_csl	policy_deductable	insured_sex	insure
0	328	48	OH	250/500	1000	MALE	
1	228	42	IN	250/500	2000	MALE	
2	134	29	OH	100/300	2000	FEMALE	
3	256	41	IL	250/500	2000	FEMALE	
4	228	44	IL	500/1000	1000	MALE	

Check the percentage of null value

In [5]:

```
# Calculate the percentage of null values in each column
null_percentage = (df.isnull().sum() / len(df)) * 100

# Display the result
print("Percentage of null values in each column:")
print(null_percentage)
```

```
Percentage of null values in each column:
months_as_customer             0.0
age                            0.0
policy_state                   0.0
policy_csl                     0.0
policy_deductable              0.0
insured_sex                    0.0
insured_education_level        0.0
insured_occupation             0.0
insured_relationship           0.0
capital-gains                  0.0
capital-loss                   0.0
incident_type                  0.0
collision_type                 0.0
incident_severity              0.0
authorities_contacted          9.1
incident_state                 0.0
incident_city                  0.0
incident_hour_of_the_day       0.0
number_of_vehicles_involved    0.0
property_damage                0.0
bodily_injuries                0.0
witnesses                      0.0
police_report_available        0.0
total_claim_amount             0.0
injury_claim                   0.0
property_claim                 0.0
vehicle_claim                  0.0
fraud_reported                 0.0
dtype: float64
```

Replace all of the "?" with "Unknown"

In [6]:

```
df.replace("?", "Unknown", inplace=True)
```

In [8]:

```
# Calculate the percentage of "Unknown" values in each column
unknown_percentage = (df == "Unknown").sum() / len(df) * 100

# Filter columns with "Unknown" values
unknown_columns = unknown_percentage[unknown_percentage > 0]

# Print the percentage of "Unknown" values in each column
print("Percentage of Unknown values in each column:")
print(unknown_percentage[unknown_columns.index])
```

```
Percentage of Unknown values in each column:
collision_type             17.8
property_damage            36.0
police_report_available    34.3
dtype: float64
```

In [9]:

```
# Drop property_damage and police_report_available
df.drop(columns=['property_damage', 'police_report_available'], inplace=True
df.head()
```

Out[9]:

	months_as_customer	age	policy_state	policy_csl	policy_deductable	insured_sex	insure
0	328	48	OH	250/500	1000	MALE	
1	228	42	IN	250/500	2000	MALE	
2	134	29	OH	100/300	2000	FEMALE	
3	256	41	IL	250/500	2000	FEMALE	
4	228	44	IL	500/1000	1000	MALE	

Check the number of Unique value for each 'Object' datatype column

In [10]:

```
# Select columns with 'object' datatype
object_columns = df.select_dtypes(include=['object'])

# Calculate the number of unique values for each 'object' datatype column
unique_value_counts = object_columns.apply(lambda x: x.nunique())

# Print the number of unique values for each column
print("Number of unique values for each 'object' datatype column:")
print(unique_value_counts)
```

```
Number of unique values for each 'object' datatype column:
policy_state                3
policy_csl                  3
insured_sex                 2
insured_education_level     7
insured_occupation         14
insured_relationship        6
incident_type               4
collision_type              4
incident_severity           4
authorities_contacted       4
incident_state              7
incident_city               7
fraud_reported              2
dtype: int64
```

Print all of the Unique value for each object datatype

In [11]:

```
# Print all unique values for each 'object' datatype column
for column in object_columns.columns:
    unique_values = df[column].unique()
    print(f"Unique values for column '{column}':")
    print(unique_values)
    print()
```

```
Unique values for column 'policy_state':
['OH' 'IN' 'IL']

Unique values for column 'policy_csl':
['250/500' '100/300' '500/1000']

Unique values for column 'insured_sex':
['MALE' 'FEMALE']

Unique values for column 'insured_education_level':
['MD' 'PhD' 'Associate' 'Masters' 'High School' 'College' 'JD']

Unique values for column 'insured_occupation':
['craft-repair' 'machine-op-inspct' 'sales' 'armed-forces' 'tech-support'
 'prof-specialty' 'other-service' 'priv-house-serv' 'exec-managerial'
 'protective-serv' 'transport-moving' 'handlers-cleaners' 'adm-clerical'
 'farming-fishing']

Unique values for column 'insured_relationship':
['husband' 'other-relative' 'own-child' 'unmarried' 'wife' 'not-in-famil
y']

Unique values for column 'incident_type':
['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']

Unique values for column 'collision_type':
['Side Collision' 'Unknown' 'Rear Collision' 'Front Collision']

Unique values for column 'incident_severity':
['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']

Unique values for column 'authorities_contacted':
['Police' nan 'Fire' 'Other' 'Ambulance']

Unique values for column 'incident_state':
['SC' 'VA' 'NY' 'OH' 'WV' 'NC' 'PA']

Unique values for column 'incident_city':
['Columbus' 'Riverwood' 'Arlington' 'Springfield' 'Hillsdale' 'Northbend'
 'Northbrook']

Unique values for column 'fraud_reported':
['Y' 'N']
```

Label Encoding

In [12]:

```
# Manually encode ordinal columns
df['policy_csl'] = df['policy_csl'].replace({'250/500': 0, '100/300': 1, '5

df['insured_education_level'] = df['insured_education_level'].replace({
    'High School': 0,
    'Associate': 1,
    'College': 2,
    'Masters': 3,
    'PhD': 4,
    'MD': 5,
    'JD': 6
})

df['incident_severity'] = df['incident_severity'].replace({
    'Trivial Damage': 0,
    'Minor Damage': 1,
    'Major Damage': 2,
    'Total Loss': 3
})

# Display the updated DataFrame
df.head()
```

Out[12]:

	months_as_customer	age	policy_state	policy_csl	policy_deductable	insured_sex	insure
0	328	48	OH	0	1000	MALE	
1	228	42	IN	0	2000	MALE	
2	134	29	OH	1	2000	FEMALE	
3	256	41	IL	0	2000	FEMALE	
4	228	44	IL	2	1000	MALE	

In [13]:

```
# Convert columns to integer datatype
df['policy_csl'] = df['policy_csl'].astype(int)
df['insured_education_level'] = df['insured_education_level'].astype(int)
df['incident_severity'] = df['incident_severity'].astype(int)
```

In [14]:

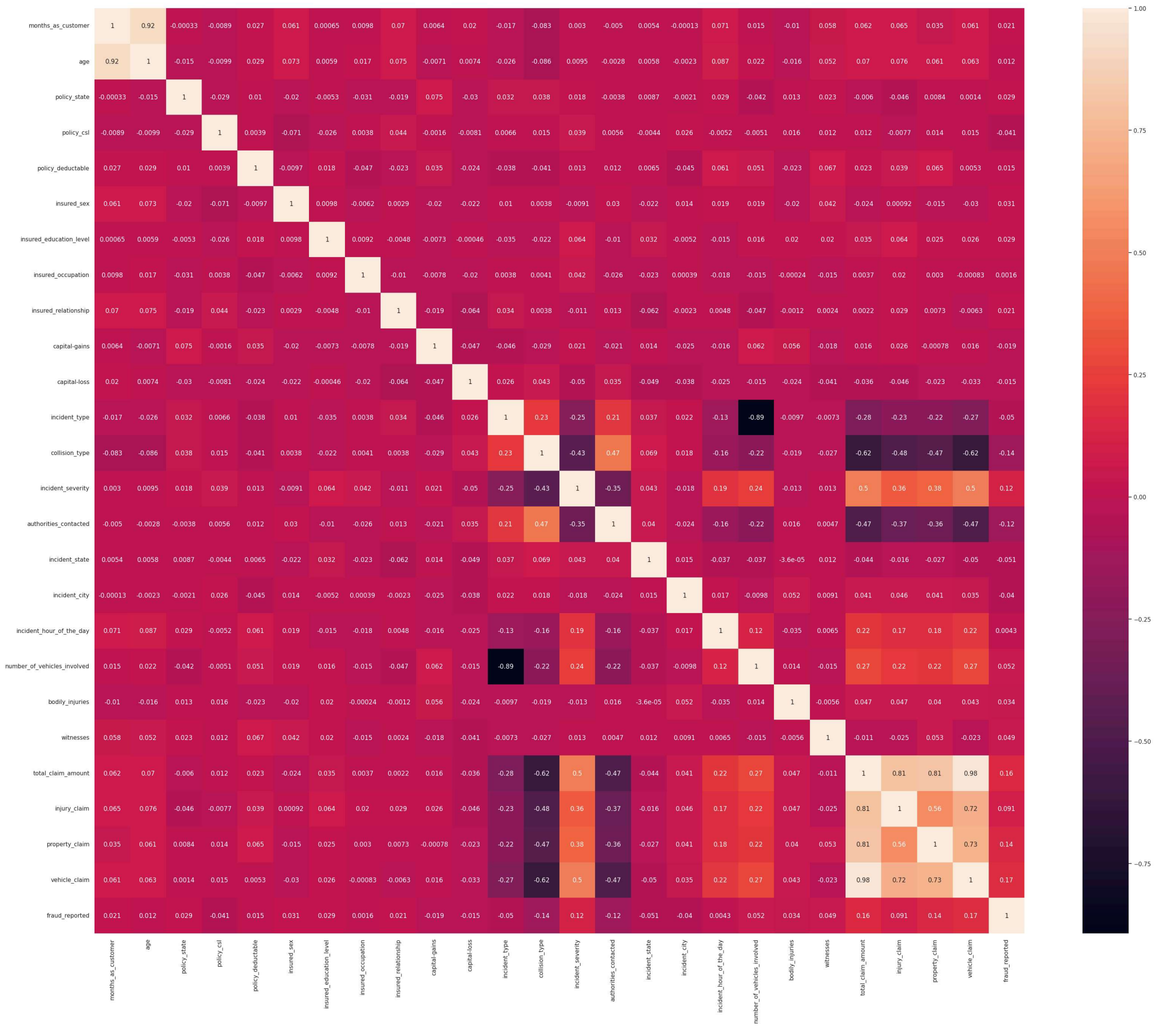
```python
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
policy_state: [2 1 0]
insured_sex: [1 0]
insured_occupation: [ 2  6 11  1 12  9  7  8  3 10 13  5  0  4]
insured_relationship: [0 2 3 4 5 1]
incident_type: [2 3 0 1]
collision_type: [2 3 1 0]
authorities_contacted: [3 4 1 2 0]
incident_state: [4 5 1 2 6 0 3]
incident_city: [1 5 0 6 2 3 4]
fraud_reported: [1 0]
```

Check the correlation between each columns

In [15]:
```
# Correlation Heatmap
plt.figure(figsize=(40, 32))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[15]:
```
<Axes: >
```



Drop column that have high correlation toreduce redudancy

In [16]:
```
df.drop(columns = ['months_as_customer', 'injury_claim',
                   'property_claim', 'vehicle_claim'], inplace=True)
df.head()
```

Out[16]:

	age	policy_state	policy_csl	policy_deductable	insured_sex	insured_education_level	insu
0	48	2	0	1000	1	5	
1	42	1	0	2000	1	5	
2	29	2	1	2000	0	4	
3	41	0	0	2000	0	4	
4	44	0	2	1000	1	1	

In [17]:
```
df.shape
```

Out[17]:
```
(1000, 22)
```

MinMax Scaler for dataframe

In [18]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the DataFrame
scaled_df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

# Replace the original DataFrame with the scaled DataFrame
df = scaled_df

# Display the scaled DataFrame
df.head()
```

Out[18]:

	age	policy_state	policy_csl	policy_deductable	insured_sex	insured_education_level
0	0.644444	1.0	0.0	0.333333	1.0	0.833333
1	0.511111	0.5	0.0	1.000000	1.0	0.833333
2	0.222222	1.0	0.5	1.000000	0.0	0.666667
3	0.488889	0.0	0.0	1.000000	0.0	0.666667
4	0.555556	0.0	1.0	0.333333	1.0	0.166667

Train Test Split

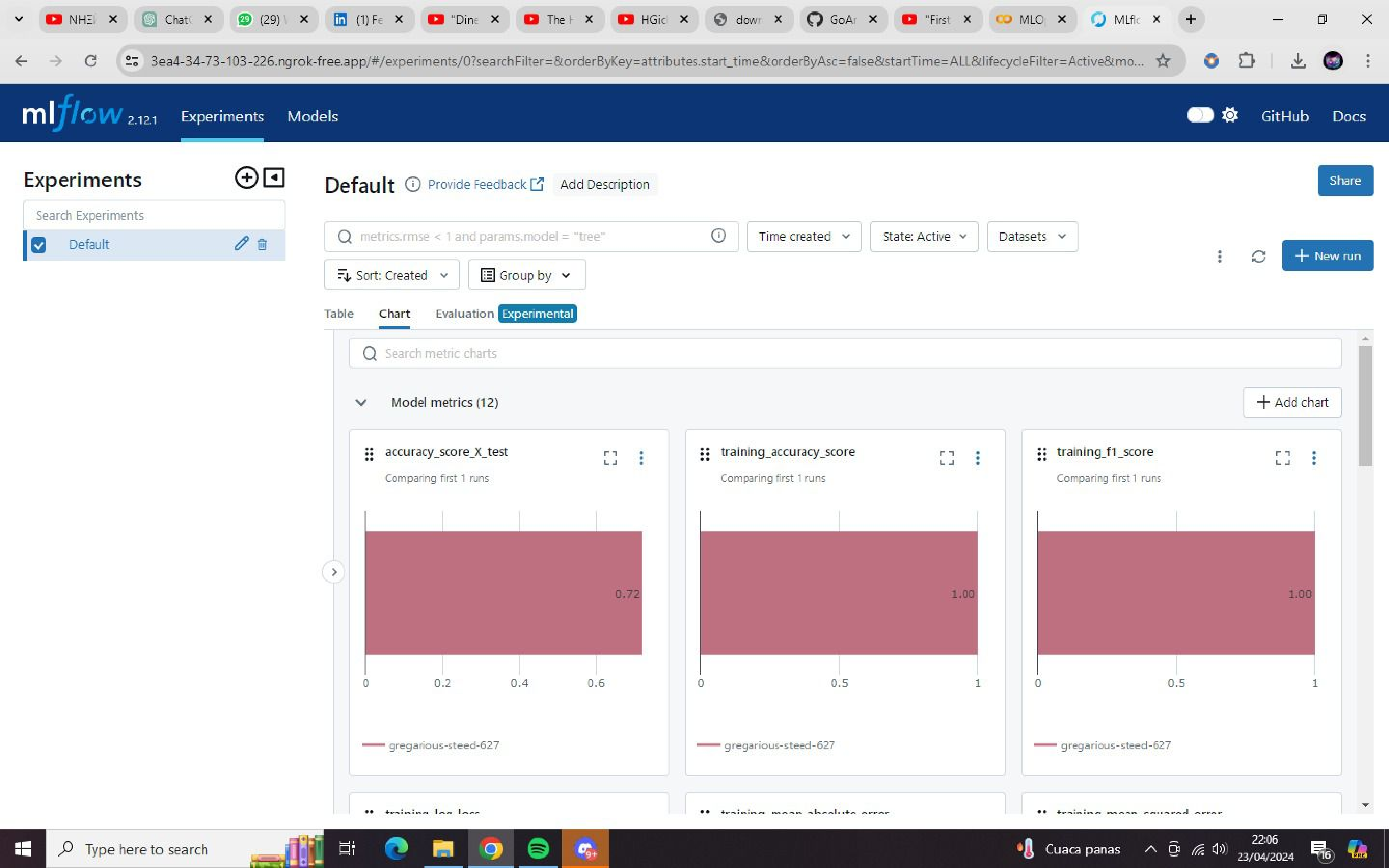
In [19]:
```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['fraud_reported'])  # Features
y = df['fraud_reported']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

In [23]:
```
import mlflow
import mlflow.sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# 1. Model Training
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# 2. Model Evaluation
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# 3. MLflow Integration
mlflow.set_tracking_uri("your_mlflow_tracking_uri")  # Set MLflow tracking U
mlflow.set_experiment("Random Forest Classifier")   # Set the experiment nam

# 4. Experiment Tracking
with mlflow.start_run():
    # Log parameters
    mlflow.log_param("n_estimators", rf_classifier.n_estimators)
    mlflow.log_param("random_state", rf_classifier.random_state)

    # Log metrics
    mlflow.log_metric("accuracy", accuracy)

    # Log model
    mlflow.sklearn.log_model(rf_classifier, "random_forest_model")
```

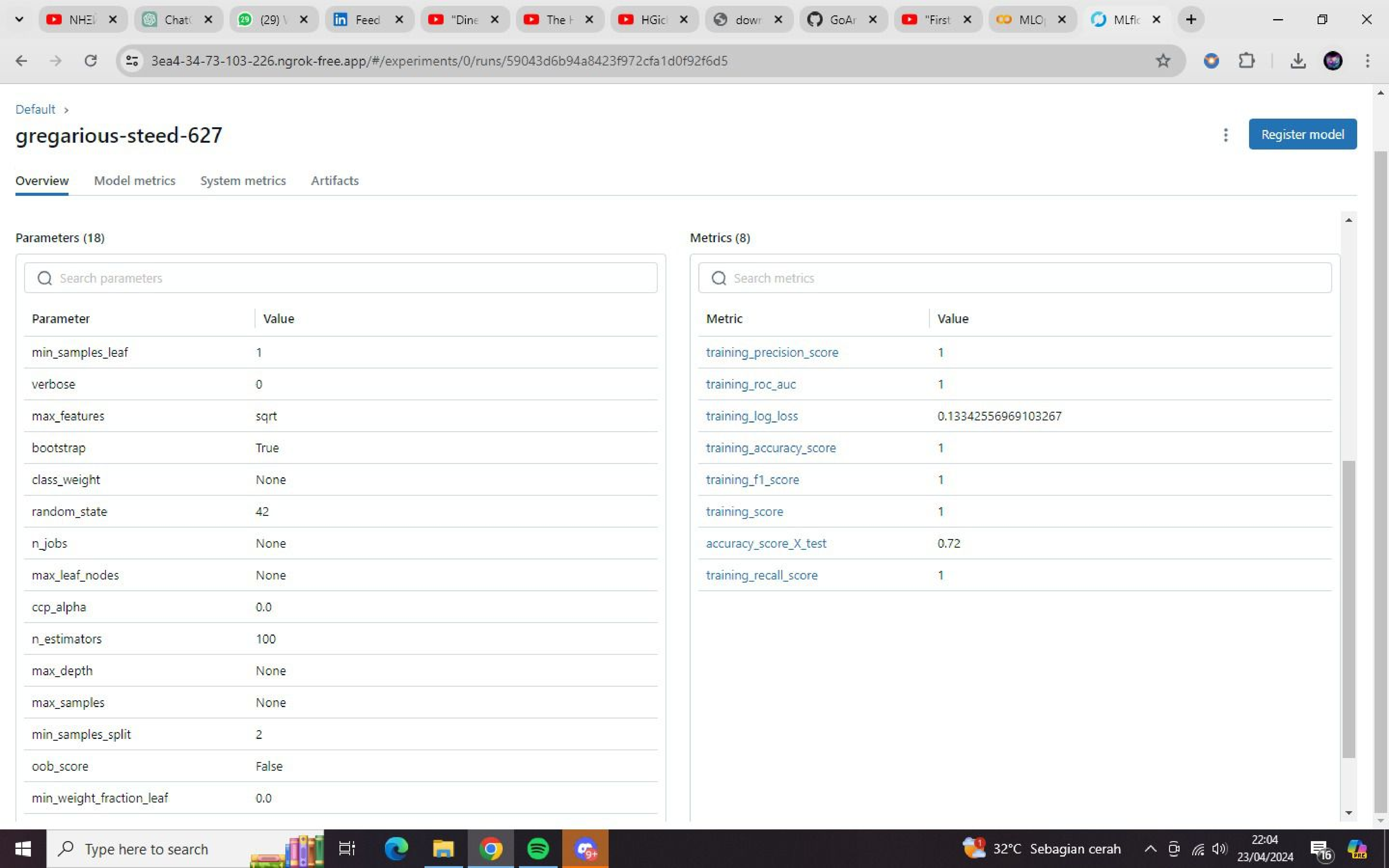
```
2024/04/24 04:52:41 INFO mlflow.utils.autologging_utils: Created MLflow au
tologging run with ID 'c44347a0d3f64d9ca1e08e269a14f704', which will track
hyperparameters, performance metrics, model artifacts, and lineage informa
tion for the current sklearn workflow
```

In [27]:
```
from pyngrok import ngrok

# Set your Ngrok authentication token
ngrok.set_auth_token("2fX5GbnoQw5s0K2bdXiEy6xFAVo_2osmY8ZJxNJx5L8ACYKuW")

# Start MLflow tracking server in the background
get_ipython().system_raw("mlflow ui --port 5000 &")

# Create a local tunnel to access MLflow UI
ngrok.connect(5000)
```

Out[27]:
```
<NgrokTunnel: "https://0fcc-34-73-103-226.ngrok-free.app" -> "http://local
host:5000">
```



3ea4-34-73-103-226.ngrok-free.app/#/experiments/0?searchFilter=&orderByKey=attributes.start_time&orderByAsc=false&startTime=ALL&lifecycleFilter=Active&mo...
mlflow 2.12.1
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
Default
Provide Feedback
Add Description
Share
metrics.rmse < 1 and params.model = "tree"
Time created
State: Active
Datasets
New run
Sort: Created
Group by
Table
Chart
Evaluation
Experimental
Search metric charts
Model metrics (12)
Add chart
accuracy_score_X_test
Comparing first 1 runs
0.72
0
0.2
0.4
0.6
gregarious-steed-627
training_accuracy_score
Comparing first 1 runs
1.00
0
0.5
1
gregarious-steed-627
training_f1_score
Comparing first 1 runs
1.00
0
0.5
1
gregarious-steed-627
training_log_loss
training_mean_absolute_error
training_mean_squared_error
Type here to search
Cuaca panas
22:06
23/04/2024



Default ›

gregarious-steed-627

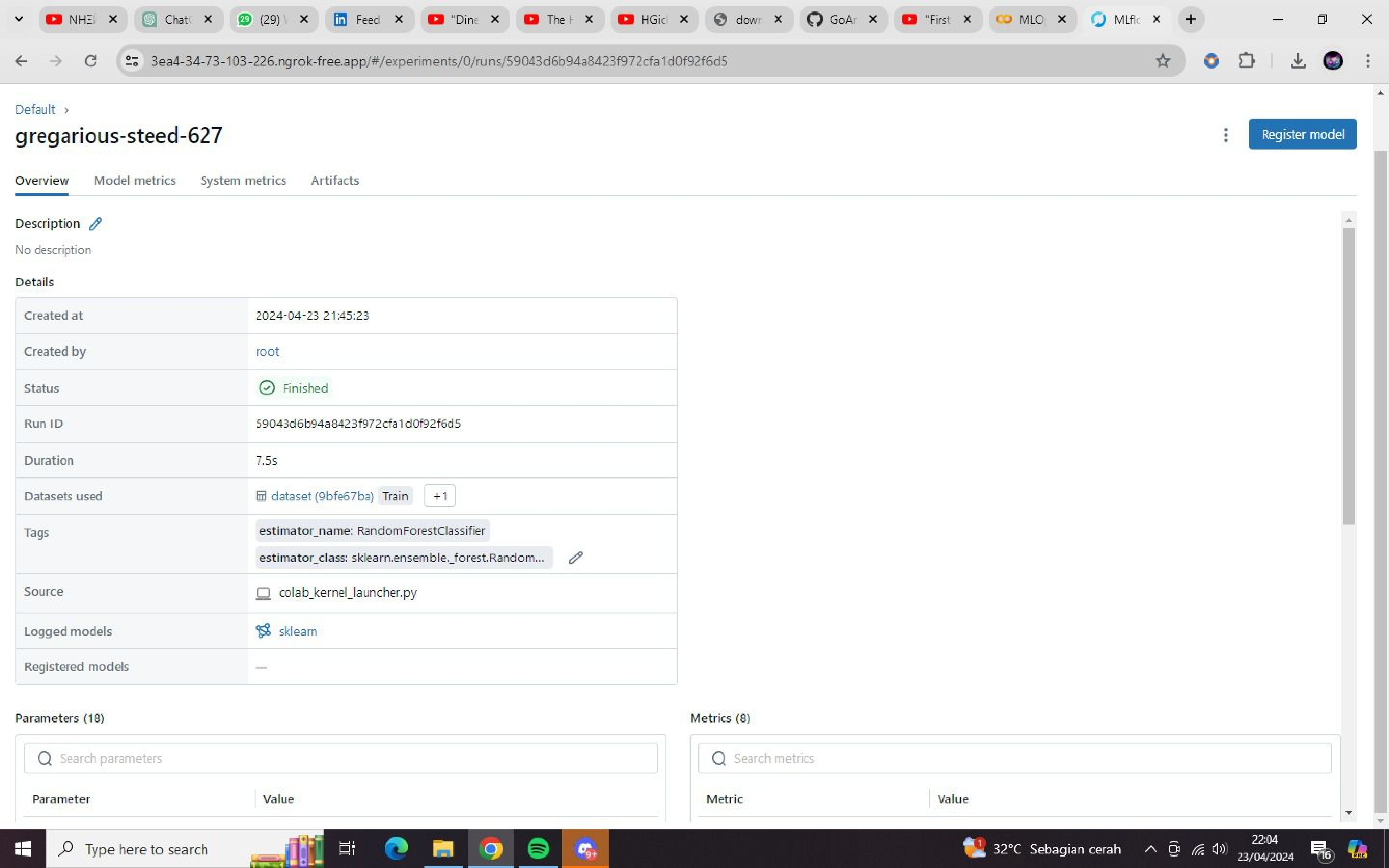
Overview | Model metrics | System metrics | Artifacts

Parameters (18)

Parameter	Value
min_samples_leaf	1
verbose	0
max_features	sqrt
bootstrap	True
class_weight	None
random_state	42
n_jobs	None
max_leaf_nodes	None
ccp_alpha	0.0
n_estimators	100
max_depth	None
max_samples	None
min_samples_split	2
oob_score	False
min_weight_fraction_leaf	0.0

Metrics (8)

Metric	Value
training_precision_score	1
training_roc_auc	1
training_log_loss	0.13342556969103267
training_accuracy_score	1
training_f1_score	1
training_score	1
accuracy_score_X_test	0.72
training_recall_score	1



Default ›

gregarious-steed-627

Description

No description

Details

Created at	2024-04-23 21:45:23
Created by	root
Status	Finished
Run ID	59043d6b94a8423f972cfa1d0f92f6d5
Duration	7.5s
Datasets used	dataset (9bfe67ba) Train +1
Tags	estimator_name: RandomForestClassifier estimator_class: sklearn.ensemble._forest.Random...
Source	colab_kernel_launcher.py
Logged models	sklearn
Registered models	—

Parameters (18)

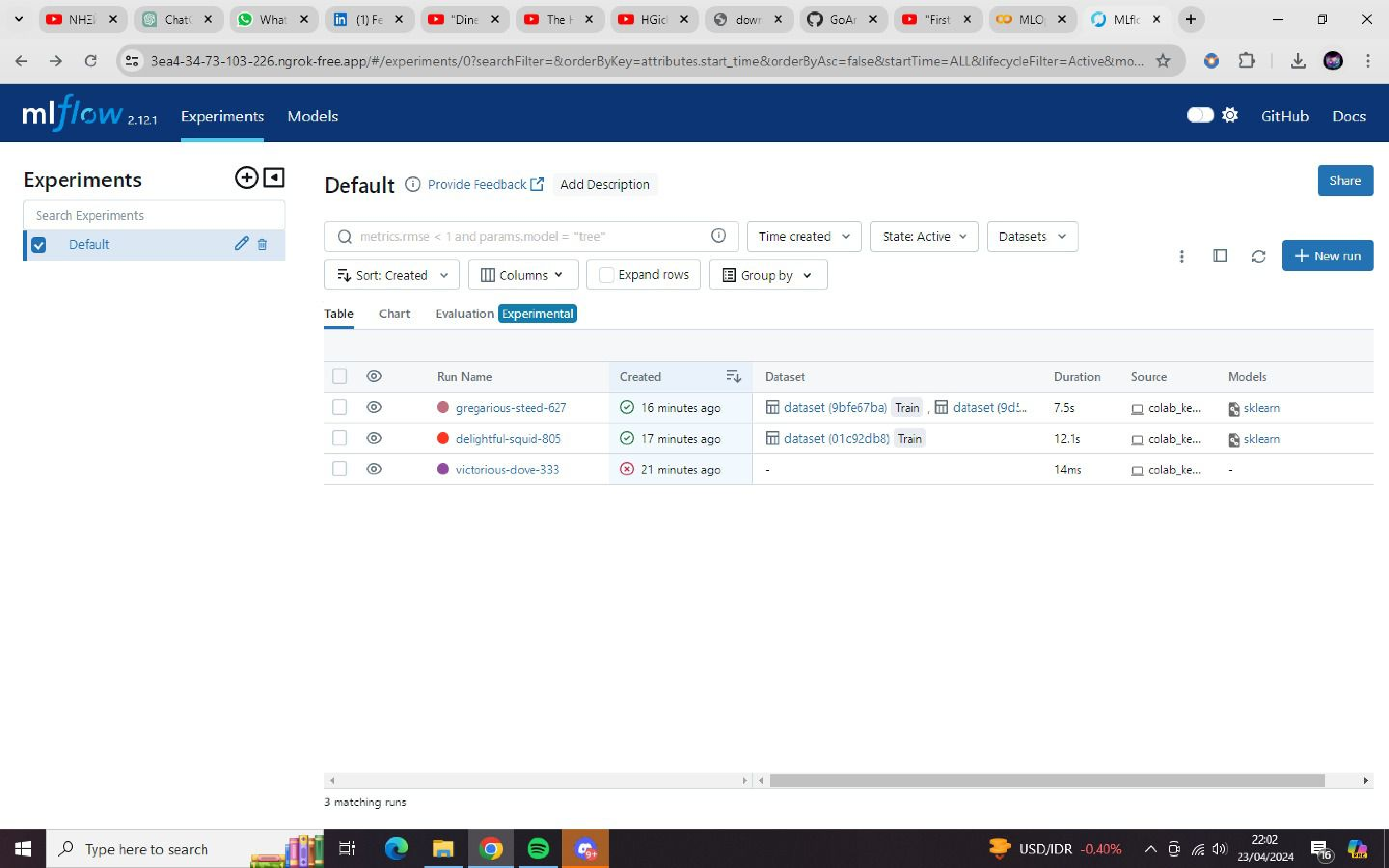
Search parameters

Parameter	Value

Metrics (8)

Search metrics

Metric	Value



3ea4-34-73-103-226.ngrok-free.app/#/experiments/0?searchFilter=&orderByKey=attributes.start_time&orderByAsc=false&startTime=ALL&lifecycleFilter=Active&mo...
mlflow 2.12.1
Experiments
Models
GitHub
Docs

Experiments

Search Experiments

☑ Default

Default ⓘ Provide Feedback Add Description

Share

metrics.rmse < 1 and params.model = "tree"

Time created | State: Active | Datasets

Sort: Created | Columns | □ Expand rows | Group by

+ New run

Table | Chart | Evaluation Experimental

□	Run Name	Created	Dataset	Duration	Source	Models
□	gregarious-steed-627	✓ 16 minutes ago	dataset (9bfe67ba) Train , dataset (9d...	7.5s	colab_ke...	sklearn
□	delightful-squid-805	✓ 17 minutes ago	dataset (01c92db8) Train	12.1s	colab_ke...	sklearn
□	victorious-dove-333	⊗ 21 minutes ago	-	14ms	colab_ke...	-

3 matching runs