MLOps

Machine Learning Operations is referred to as MLOps. Streamlining the process of putting machine learning models into production, then maintaining and monitoring them, is the basic task of machine learning engineering, or MLOps. MLOps is a team effort that frequently includes data scientists, devops engineers, and IT.

Why are MLOps necessary?

Machine learning is challenging to commercialize. Data ingest, data preparation, model training, model tuning, model deployment, model monitoring, explain ability, and many other complicated elements make up the machine learning lifecycle. Additionally, it necessitates cross-team cooperation and handoffs from the Data Engineering, Data Science, and ML Engineering teams. Naturally, maintaining synchronization and coordination across all of these activities calls for rigorous operational rigor. The machine learning lifecycle's experimentation, iteration, and continual improvement are all included in MLOps.

What parts of MLOps are there?

In machine learning projects, the MLOp range can be as narrow or broad as the project requires. While in some projects MLOps implementation may be limited to the model deployment process, in other projects it may be necessary to include everything from the data pipeline through model production. Most businesses use MLOps concepts in the following areas:

- Exploratory data analysis (EDA)
- Data Prep and Feature Engineering
- Model training and tuning
- Model review and governance
- Model inference and serving
- Model monitoring
- Automated model retraining

MLOps Methodology

- Model Serialization and Deserialization
- Application Integration (Using Streamlit)
- Experiment Tracking and Model Management
- Orchestrate ML Pipeline
- Deployment of Ml model (Using Heroku)

Model Serialization and Deserialization

Serialization

The process of Serialization involves changing the overall data structure of the trained model into adaptable retrieval formats in order to make it easier to subsequently decompose the serialised model into production. Pickling is the generic name for using the pickle module, which involves converting the learned model parameters into a byte stream and then deserializing them at the production end.

Serialization

```
In [80]: from pickle import dump

         dump(scaler, open(r'C:\Users\Dell\del\models\standard_scaler.pkl', 'wb'))
         dump(knn, open(r'C:\Users\Dell\del\models\knnregression.pkl', 'wb'))
         dump(linear_model, open(r'C:\Users\Dell\del\models\linearregression.pkl', 'wb'))
         dump(RF, open(r'C:\Users\Dell\del\models\randomforestregression.pkl', 'wb'))
```

Deserialization

When using the Scikit Learn library for machine learning, we must save the trained models in a file and then restore them so that we can use them again to compare the model to other models and test the model on new data. Data saving is referred to as serialization, and data retrieval as deserialization.

Deserialization

```
In [81]: from pickle import load

In [82]: RF = load(open(r'C:\Users\Dell\del\models\randomforestregression.pkl', 'rb'))
         scaler = load(open(r'C:\Users\Dell\del\models\standard_scaler.pkl', 'rb'))
```

Application Integration (Using Streamlit)

app.py

C: > Users > Dell > del > app > app.py > ...

```python
import streamlit as st
import numpy as np
import pandas as pd
from pickle import load

st.title("Predict the diamond price")

# here we define some of the front end elements of the web page like
# the font and background color, the padding and the text to be displayed
html_temp = """
    <div style ="background-color:black;padding:13px">
    <h1 style ="color:white;text-align:center;"> Machine learning Application for  Predicting the Diamond Price  </h1>
    </div>
    <h2> Enter the details about your diamond to know its price </h2>

    """

# this line allows us to display the front end aspects we have
# defined in the above code
st.markdown(html_temp, unsafe_allow_html=True)

RF = load(open(r'C:\Users\Dell\del\app\randomforestregression.pkl', 'rb'))

scaler = load(open(r'C:\Users\Dell\del\app\standard_scaler.pkl', 'rb'))

label_cut = {'Ideal':2, 'Premium':3, 'Very Good':4, 'Good':1, 'Fair':0}
label_color = {'G':3, 'E':1, 'F':2, 'H':4, 'D':0, 'I':5, 'J':6}
label_clarity = {'SI1':2, 'VS2':5, 'SI2':3, 'VS1': 4, 'VVS2':7, 'VVS1':6, 'IF':1, 'I1':0}
carat = st.slider("Carat", 0.2, 5.01)
table = st.slider("Width", 43.00, 95.00)
depth = st.slider("Depth", 43.00, 79.00)
x = st.slider('X',0.00,10.70)
y = st.slider('Y',0.00,58.90 )
z = st.slider('Z',0.00,31.80 )

cut = st.selectbox(
    'How would be the cut of Diamond?',
```

Making the application using streamlit library of the python

```python
cut = st.selectbox(
    'How would be the cut of Diamond?',
    ('Fair', 'Good', 'Very Good', 'Ideal', 'Premium'))

color = st.selectbox(
    'What should be the color of Diamond?',
    ('J', 'I', 'H', 'G', 'F', 'E', 'D'))

clarity = st.selectbox(
    'How would you like to be contacted?',
    ('I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF'))

btn_click = st.button("Predict")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
PS C:\Users\Dell> & C:/Users/Dell/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Dell/del/app/app.py
2022-09-28 21:47:31.653
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run c:/Users/Dell/del/app/app.py [ARGUMENTS]
2022-09-28 21:47:36.217 Session state does not function when running a script without `streamlit run`
PS C:\Users\Dell> streamlit run c:/Users/Dell/del/app/app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.4:8501
```

Running the application on local host by using the command streamlit run app.py

So after putting all the values in the application and clicking the predict button we get the price of our diamond. As you can see in application our diamond price comes out to be $3283.33

Experiment Tracking and Model Management

Introduction to Experiment Tracking

Experiment tracking is the practice of collecting all experiment-related information for each experiment that you execute.

Things we have to track for each Experiment Run?

1. Training and Validation Data Used
2. Hyper parameters
3. Metrics
4. Models

When models are put into production, ML model management begins:

- Streamlines the process of moving models from experimentation to production
- Aids in model versioning
- Arranges model artefacts in an ML model registry
- Aids in testing different model versions in the production environment
- Allows rolling back to an earlier model version if the more recent one appears to be having problems.

Even if your models are not used in production, experiment tracking is still valuable (yet). And they might never get there in many programmers, particularly those that are research-focused. However, knowing all of the experiment-related metadata assures that you will be prepared for this golden time.

Best Tools for ML Experiment Tracking and Management

- Neptune
- Weights & Biases
- MLflow
- TensorBoard

- ➢ Guild AI

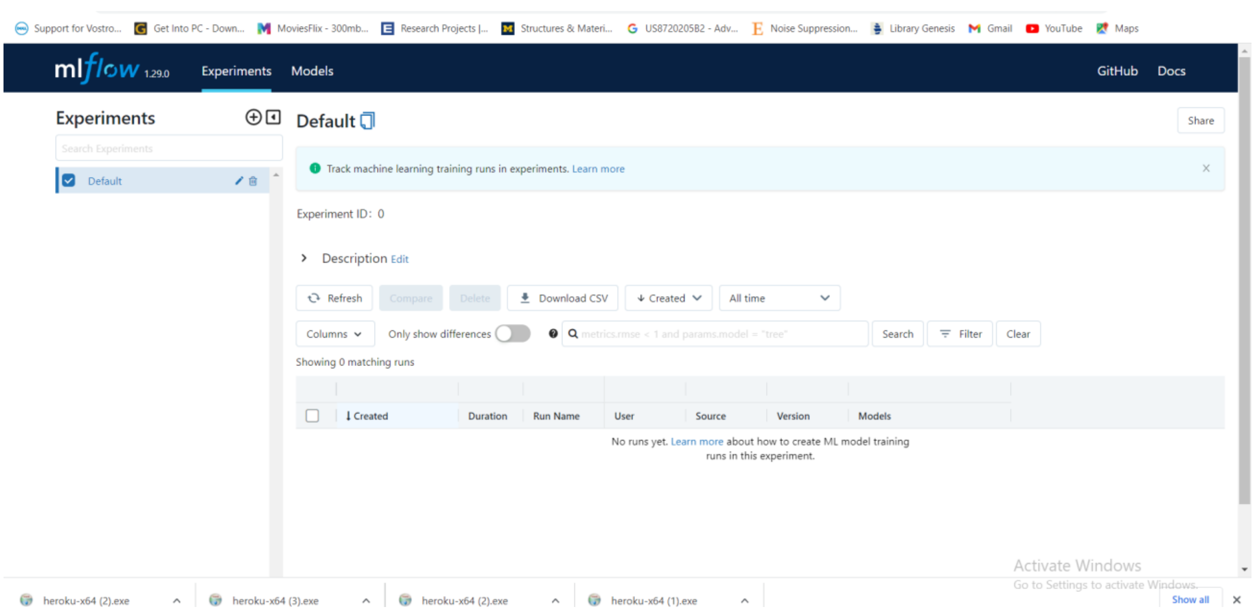
In our project we will use mlflow open source python based library

MLFlow keeps track of:

- ➢ Tags
- ➢ Parameters
- ➢ Metrics
- ➢ Models
- ➢ Artifact
- ➢ Source code, Start and End Time, Authors etc..

Run below mentioned commands to install mlflow on your system>**pip install mlflow**

To view mlflow dashboard run below command>**Mlflow ui**



Creating database file in our dic to store all the logs

Command to create sql database>`mlflow ui --backend-store-uri sqlite:///mlflow.db`

Step 1 - Import MLFlow

```
Import mlflow
```

Step 2 - Set the tracker and experiment

```
authlib-3.2.1 prometheus-flask-exporter-0.20.3 pywin32-304 querystring-parser-1.2.4 smmap-5.0.0 sqlparse-0.4.3 waitress-2.1.
```

```
In [123]: mlflow.set_tracking_uri("sqlite:///mlflow.db")

          mlflow.set_experiment("Diamond Price Prediction")
```

```
2022/09/28 11:43:55 INFO mlflow.store.db.utils: Creating initial MLflow database tables...
2022/09/28 11:43:55 INFO mlflow.store.db.utils: Updating database tables
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 451aebb31d03, add metric step
INFO  [alembic.runtime.migration] Running upgrade 451aebb31d03 -> 90e64c465722, migrate user column to tags
INFO  [alembic.runtime.migration] Running upgrade 90e64c465722 -> 181f10493468, allow nulls for metric values
INFO  [alembic.runtime.migration] Running upgrade 181f10493468 -> df50e92ffc5e, Add Experiment Tags Table
INFO  [alembic.runtime.migration] Running upgrade df50e92ffc5e -> 7ac759974ad8, Update run tags with larger limit
INFO  [alembic.runtime.migration] Running upgrade 7ac759974ad8 -> 89d4b8295536, create latest metrics table
INFO  [89d4b8295536_create_latest_metrics_table_py] Migration complete!
INFO  [alembic.runtime.migration] Running upgrade 89d4b8295536 -> 2b4d017a5e9b, add model registry tables to db
INFO  [2b4d017a5e9b_add_model_registry_tables_to_db_py] Adding registered_models and model_versions tables to database.
INFO  [2b4d017a5e9b_add_model_registry_tables_to_db_py] Migration complete!
INFO  [alembic.runtime.migration] Running upgrade 2b4d017a5e9b -> cfd24bdc0731, Update run status constraint with killed
INFO  [alembic.runtime.migration] Running upgrade cfd24bdc0731 -> 0a8213491aaa, drop_duplicate_killed_constraint
INFO  [alembic.runtime.migration] Running upgrade 0a8213491aaa -> 728d730b5ebd, add registered model tags table
INFO  [alembic.runtime.migration] Running upgrade 728d730b5ebd -> 27a6a02d2cf1, add model version tags table
INFO  [alembic.runtime.migration] Running upgrade 27a6a02d2cf1 -> 84291f40a231, add run_link to model_version
INFO  [alembic.runtime.migration] Running upgrade 84291f40a231 -> a8c4a736bde6, allow nulls for run_id
```

Step 3 - Start an experiment run

```
With mlflow.start_run():
```

Experiment 1 - Training KNN Regressor

```python
from sklearn import metrics
```

```python
with mlflow.start_run():
    mlflow.set_tag("developer","AJAYKHATRI")
    mlflow.set_tag("Algorithm","KNN")
```

Step 4 - Logging the metadata

```
mlflow.set_tag(KEY, VALUE)

mlflow.log_param(KEY, VALUE) mlflow.log_metric(KEY, VALUE)
```

```python
# log the data for each run using log_param, log_metric, log_model
mlflow.log_param("data-path","data/diamonds.csv")
k=3
mlflow.log_param("n_neighbors",k)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_transformed, y_train)
y_test_pred = knn.predict(X_test_transformed)
MAE=metrics.mean_absolute_error(y_test, y_test_pred)
MSE=metrics.mean_squared_error(y_test, y_test_pred)
RMSQ=np.sqrt(metrics.mean_squared_error(y_test, y_test_pred))
mlflow.log_metric("Mean Absolute Error",MAE)
mlflow.log_metric("Mean Squared Error",MSE)
mlflow.log_metric("Root Mean Squared Error",RMSQ)
```

Step 5 - Logging the model and other files (2 ways)

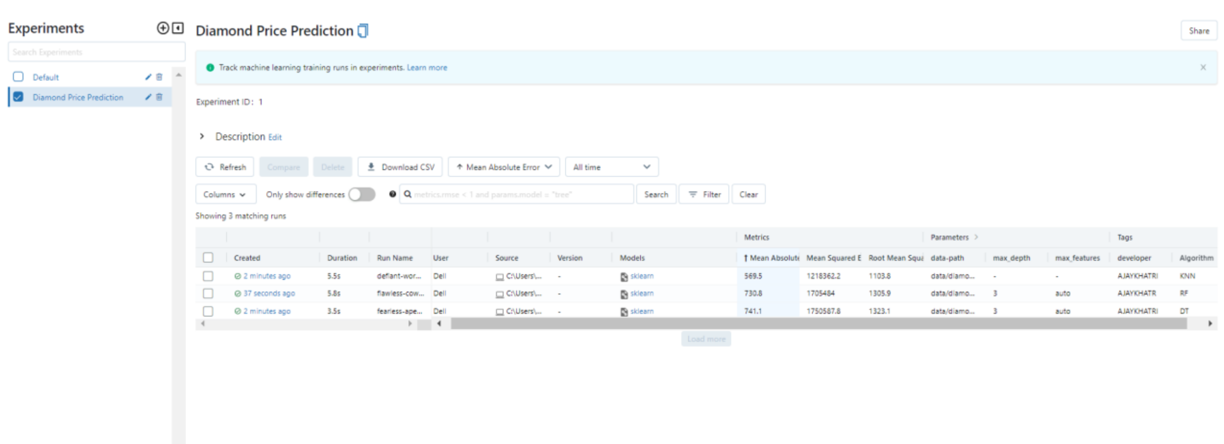
Way 1 - mlflow.<FRAMEWORK>.log_model(MODEL_OBJECT, artifact_path="PATH")

Way 2 - mlflow.log_artifact(LOCAL_PATH, artifact_path="PATH")

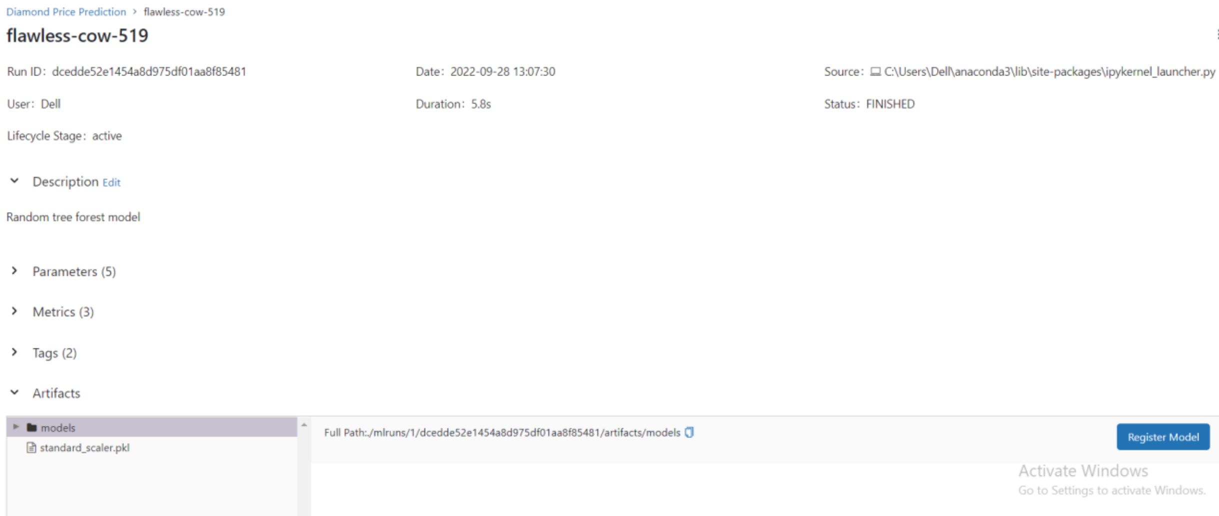
```python
mlflow.sklearn.log_model(knn,artifact_path="models")
mlflow.log_artifact("models\standard_scaler.pkl")
```

Experiment 2 - Training Decision Tree Regression

Running three different experiment (knn, decision tree, random forest and log them in sql database and will analyze with mlflow)



As is obvious, the accuracy of the random forest tree model is higher than that of the other three models. Consequently, we shall focus more on the rft model.



This is how we may access our model description and any associated parameters, metrics, tags

Random tree forest model

Parameters (5)

Name	Value
data-path	data/diamonds.csv
max_depth	3
max_features	auto
max_leaf_nodes	50
n_estimators	100

Hyper parameters for tuning the random forest tree model

- **Max_depth** represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data. We fit each decision tree with depths ranging from 1 to 32 and plot the training and test errors.

- **Max_features**- These are the most attributes that Random Forest is permitted to test in a single tree. To assign the most features, Python offers a variety of alternatives. Some of them are as follows:

  - Auto/None: This will automatically choose all the characteristics that apply to each tree. Here, we just do not impose any limitations on the specific tree.
  - Sqrt: This option will calculate the square root of each run's overall feature count. If there are 100 variables altogether, for example, we can only use 10 of them in a single tree.
  - 0.2: With this setting, the random forest may use 20% more variables throughout each run. When we wish x% of the characteristics to be taken into account, we can assign and value in the format "0.x".

- **Max_leaf_nodes**-If you've ever constructed a decision tree, you can understand the significance of the minimal sample leaf size. A decision tree's leaf node is its conclusion. The model is more susceptible to picking up noise in train data when the leaf is smaller. Generally speaking, I want minimum leaf sizes of at least 50. To identify the best option for your use case, you should test out various leaf sizes.

- **N estimators**: This refers to the number of trees you wish to construct before calculating the maximum voting or prediction averages. Although your code runs slower with more trees, it performs better. Because doing so strengthens and stabilizes your predictions, you should select the highest value your CPU can manage.

Evaluation Metrics

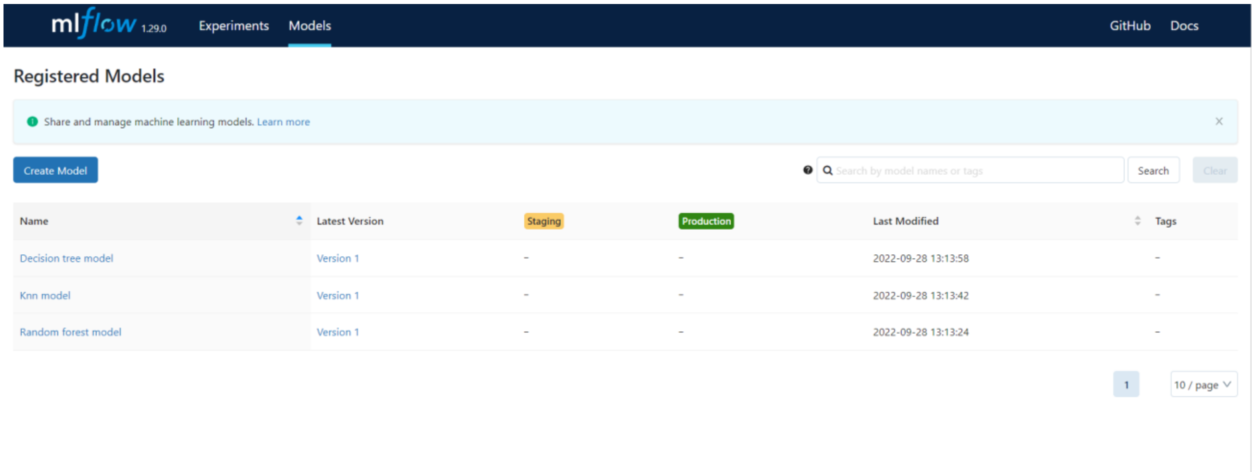
Metrics (3)

Name	Value
Mean Absolute Error	730.8
Mean Squared Error	1705484
Root Mean Squared Error	1305.9

In this tab we can see our accuracy metrics and its value

- **Mean absolute error** -One of the most basic loss functions and a simple evaluation measure is mean absolute error, often known as L1 loss. It is determined by averaging the absolute difference between the actual values and the anticipated values throughout the whole dataset.
- **Mean Squared Error (MSE** - One of the most used regression loss functions is MSE. The difference between the predicted value and the actual value is squared, averaged throughout the dataset, and used to determine the error in the mean squared error, also referred to as the L2 loss. MSE is sometimes referred to as a quadratic loss since the penalty is squared rather than directly proportional to the mistake. The outliers are given more weight when the error is squared, creating a smooth gradient for minor errors.
- **Root Mean Squared Error (RMSE)** - By calculating the square root of MSE, RMSE is calculated. Also known as the Root Mean Square Deviation, RMSE. The average error magnitude is measured, and the differences from the true value are of concern. A model has a perfect fit when the RMSE value is 0. The model and its predictions are better the smaller the RMSE. A greater RMSE denotes a significant departure between the residual and the ground truth.

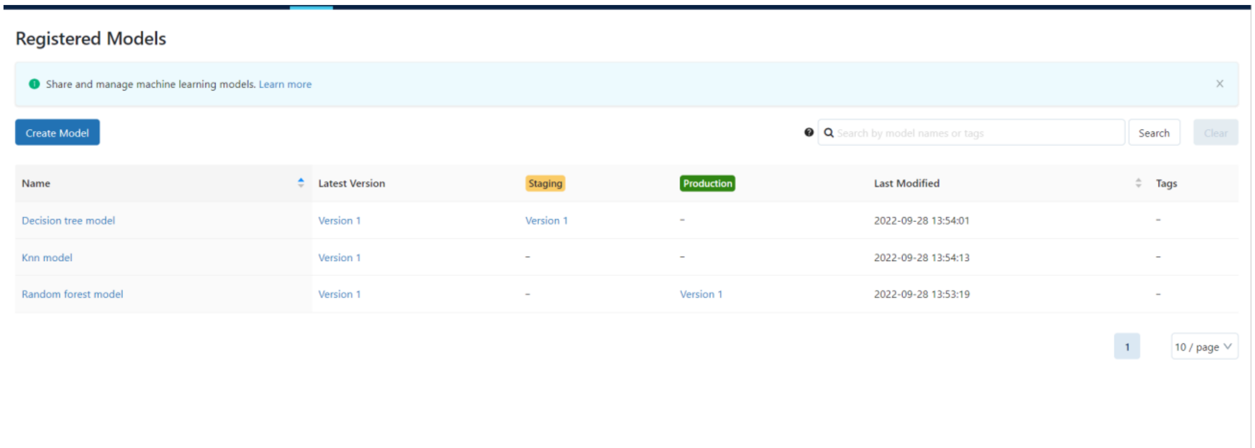
Model management



One component of MLOps is model management. All business needs should be met by ML models at scale, and they should be consistent. A clear, simple model management policy is necessary to make this happen. Development, training, versioning, and deployment of ML models are handled via ML model management.

ML Model Management components

- Version control systems assist programmers in controlling changes to source code. To manage the modifications of models in connection to datasets and vice versa, data version control is a set of tools and procedures that aims to adapt the version control process to the data world.
- Experiment Tracker: This tool is used to gather, arrange, and monitor data on model training/validation performance throughout a number of runs with various setups and datasets.
- A centralized tracking system for trained, staged, and deployed machine learning models is known as the model registry.
- Model surveillance: It is used to monitor the model's inference performance and spot any indications of Serving Skew, which occurs when data changes result in a decline in the deployed model's performance below the mark/accuracy it demonstrated in the training environment



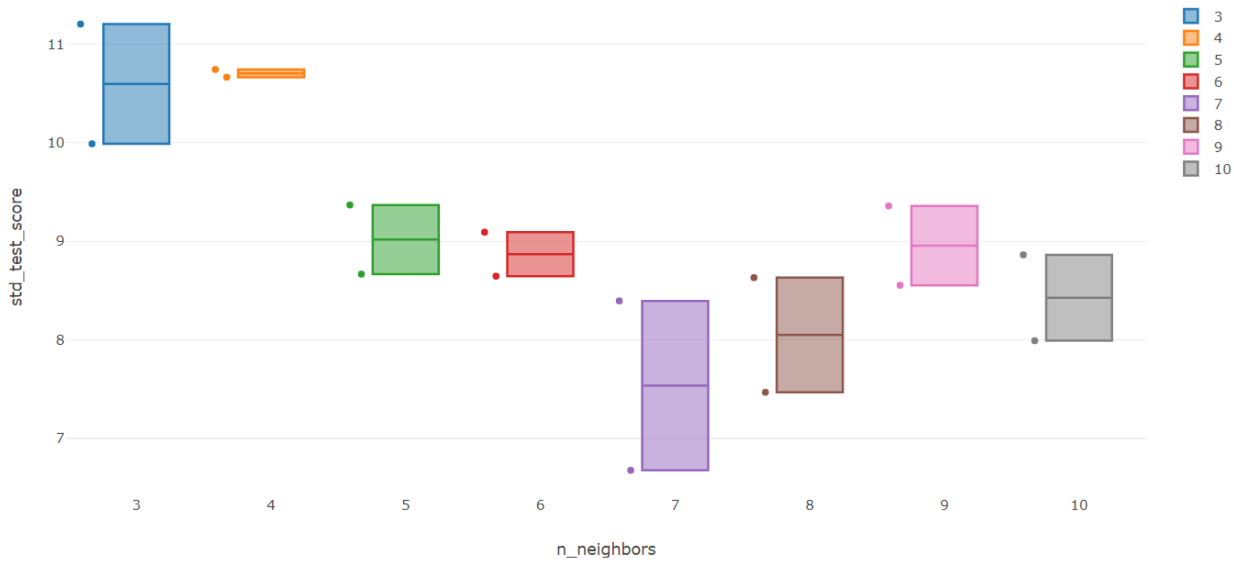
This is how manager can manage the different models according to their accuracy can use as production or staging or even archived with different version by changing the parameters

Training KNN Regressor with Hyperparameter Tuning

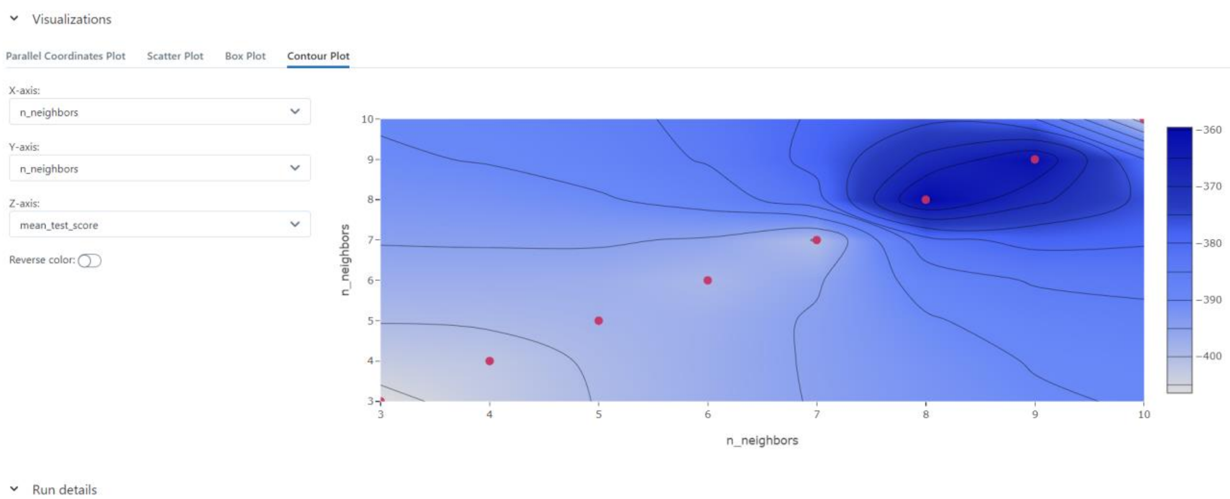
```
# Enabling automatic MLflow logging for scikit-learn runs
mlflow.sklearn.autolog(max_tuning_runs=None)

with mlflow.start_run():
    tuned_parameters = [{'n_neighbors':[i for i in range(1, 25)], 'p':[1, 2]}]

    reg = GridSearchCV(
        estimator=KNeighborsRegressor(),
        param_grid=tuned_parameters,
        scoring='neg_mean_absolute_error',
        cv=5,
        return_train_score=True,
        verbose=1
    )
    reg.fit(X_train_transformed, y_train)

    # Disabling autologging
    mlflow.sklearn.autolog(disable=True)
```

This will automatically tune my model for different k values starting from 1 to 25 and also two different p values

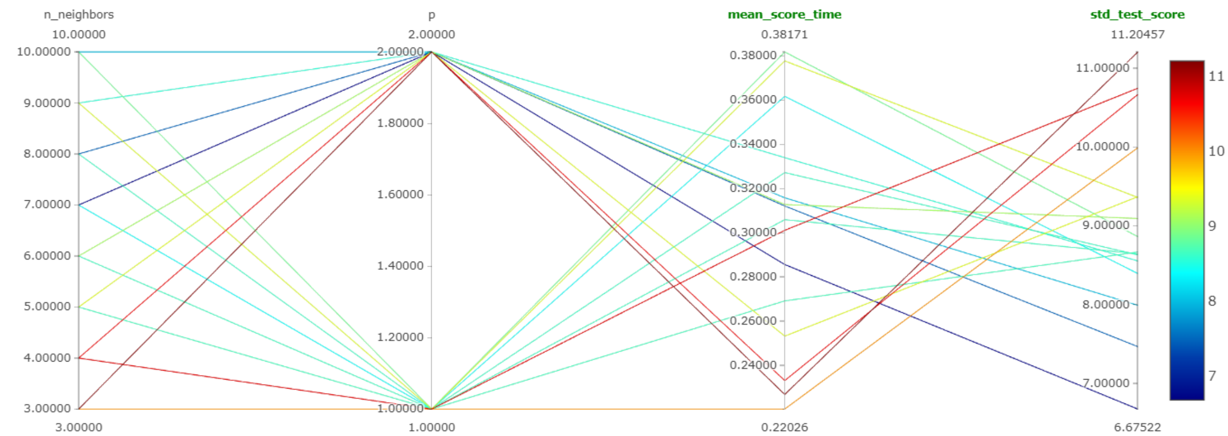
Box plot between neighbors value and std test score



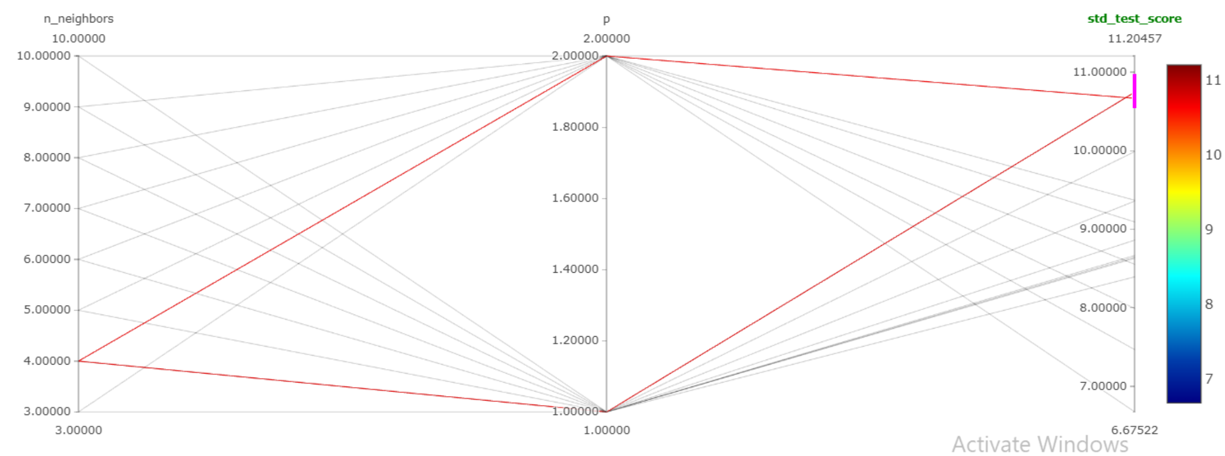
Contour plot between mean score and n_neigbors



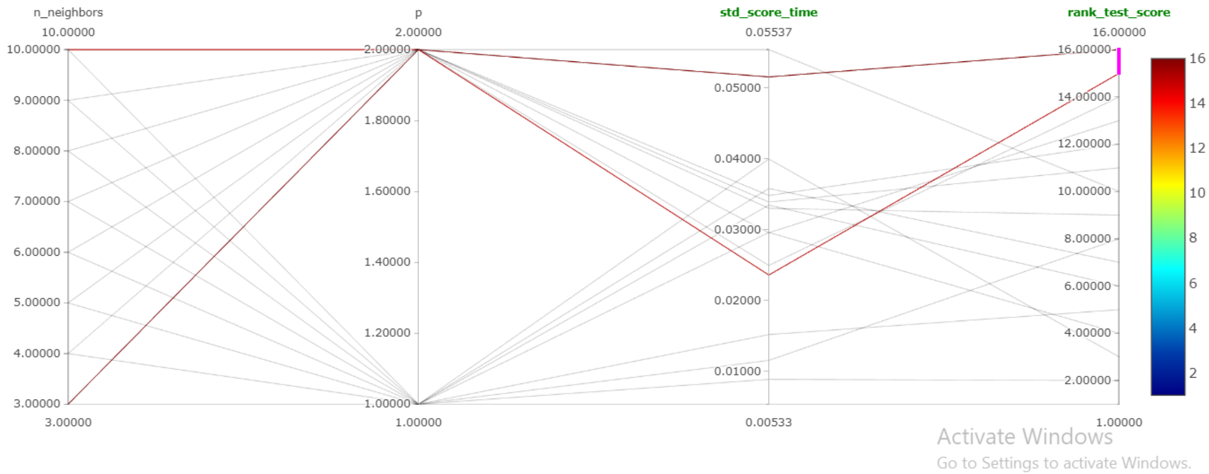
Parallel Coordinates Plot



From this contour plot now we have to optimize our hyper parameter



Now if I tune my plot with high std test score so it give me optimize parameter ad n is equal to4 and p value is equal to 1



Now in this I wanted to optimize my parameters which will takes least time to run the model and rank test score so I get two n value as 10 and 3 and p value equal to 2

Managing Machine Learning Workflows using Prefect 2.0

Why Prefect?

- Python based open source tool
- Manage ML Pipelines
- Schedule and Monitor the flow
- Gives observability into failures
- Native dask integration for scaling (Dask is used for parallel computing)

Creating and activating a Virtual Environment

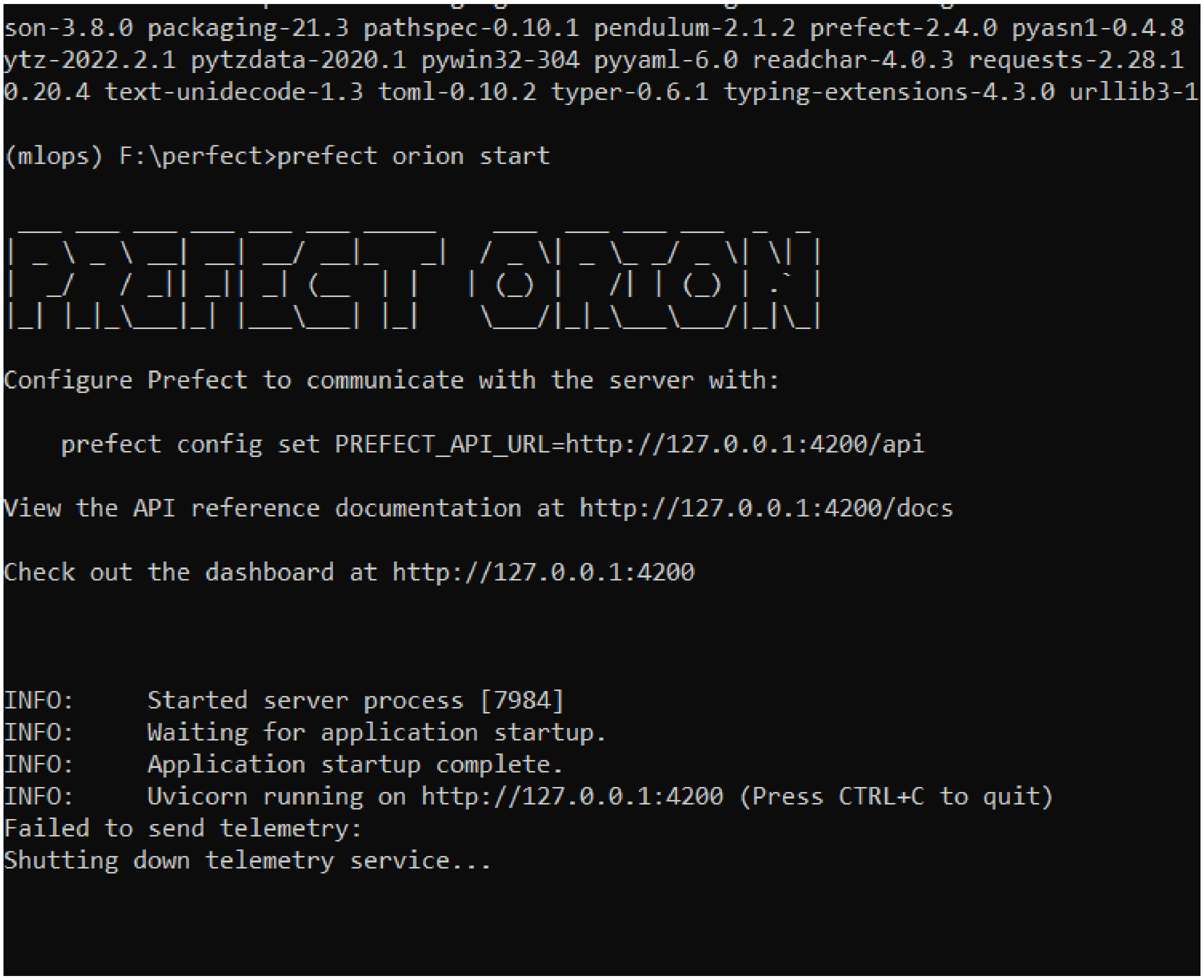
- In order to install prefect, create a virtual environment:

```
$ python -m venv mlops
```

- Enter the Virtual Environment using below mentioned command:

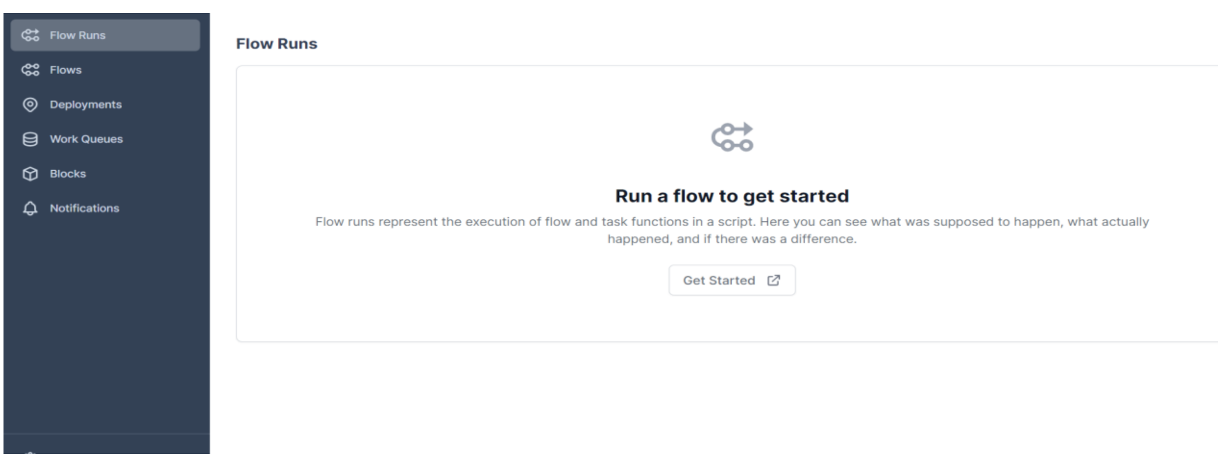
```
$ .\mlops\Scripts\activate
```

- Installing Prefect 2.0> `pip install prefect`

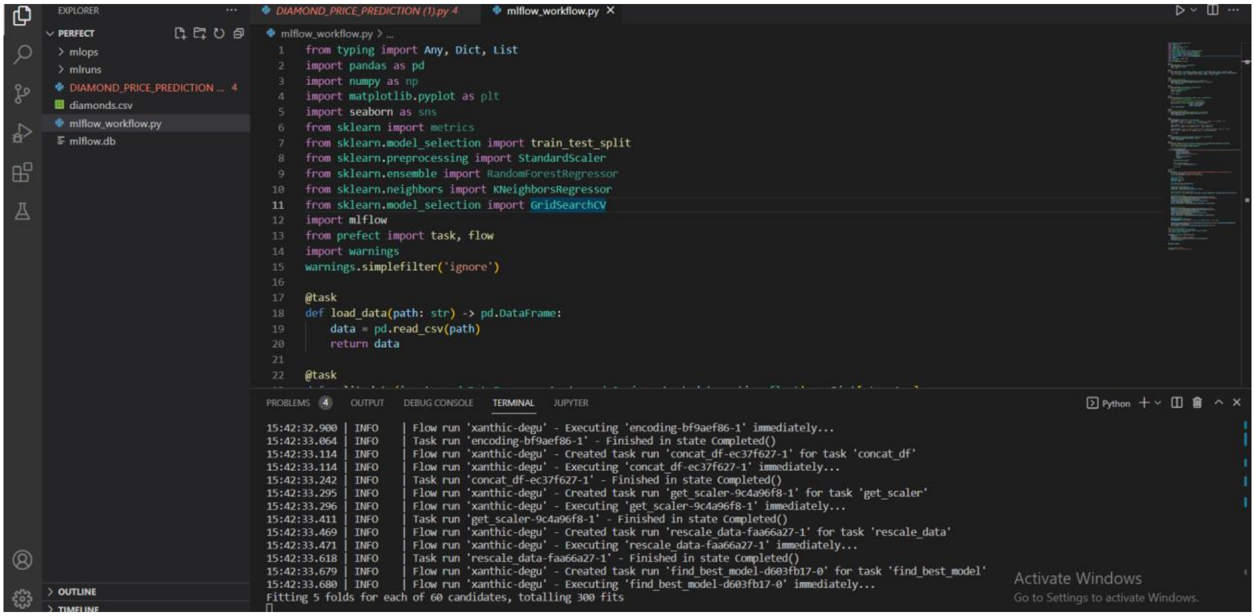


```
son-3.8.0 packaging-21.3 pathspec-0.10.1 pendulum-2.1.2 prefect-2.4.0 pyasn1-0.4.8
ytz-2022.2.1 pytzdata-2020.1 pywin32-304 pyyaml-6.0 readchar-4.0.3 requests-2.28.1
0.20.4 text-unidecode-1.3 toml-0.10.2 typer-0.6.1 typing-extensions-4.3.0 urllib3-1

(mlops) F:\perfect>prefect orion start

 ___ ___ ___ ___ ___ ___ _____    ___  ___ ___ ___  _  _
| _ \ _ \ __| __| __/ __|_   _|  / _ \| _ \_ _/ _ \| \| |
|  _/   / _|| _|| _| (__  | |   | (_) |   /| | (_) | .` |
|_| |_|_\___|_| |___\___| |_|    \___/|_|_\___\___/|_|\_|

Configure Prefect to communicate with the server with:

    prefect config set PREFECT_API_URL=http://127.0.0.1:4200/api

View the API reference documentation at http://127.0.0.1:4200/docs

Check out the dashboard at http://127.0.0.1:4200



INFO:     Started server process [7984]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:4200 (Press CTRL+C to quit)
Failed to send telemetry:
Shutting down telemetry service...
```

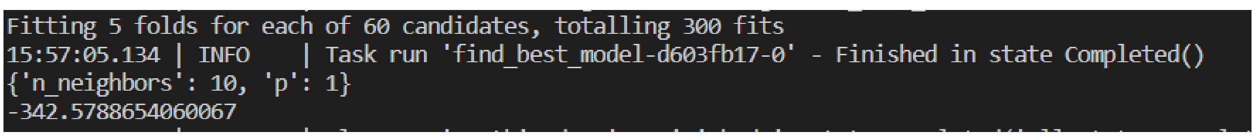
Prefect orion dashboard start

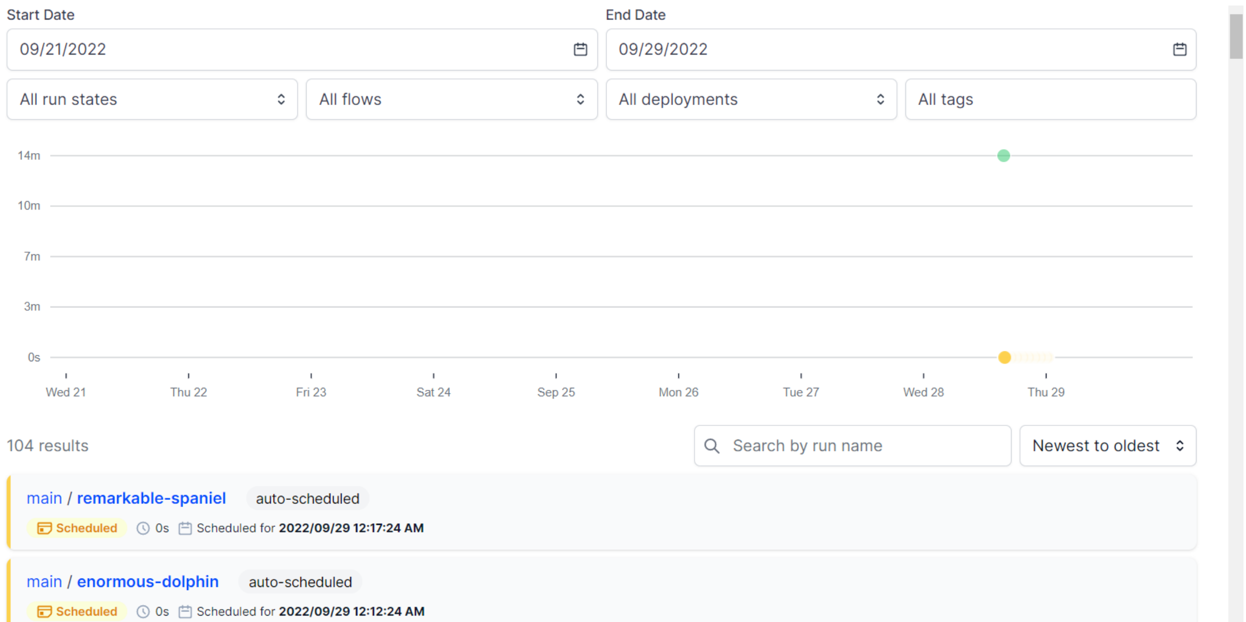


After that we have to create our workflow python script file in which we have to use opps python programming technique with all self-defined functions in it.

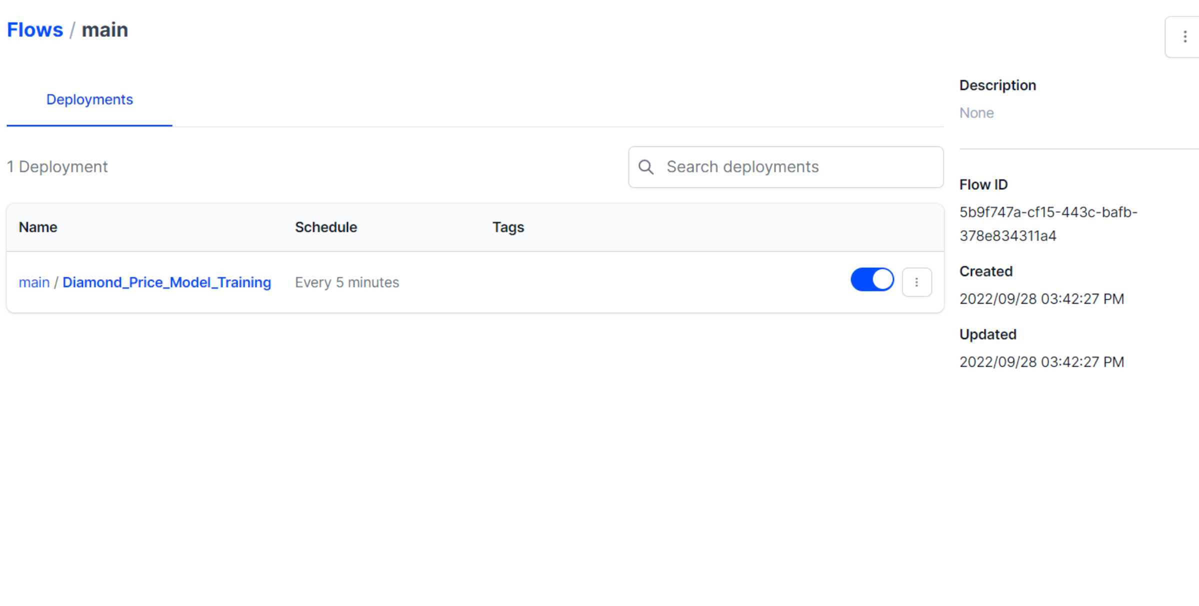


After running all the prefect orion so it gives us the optimize parameters our model and choose best model for us.

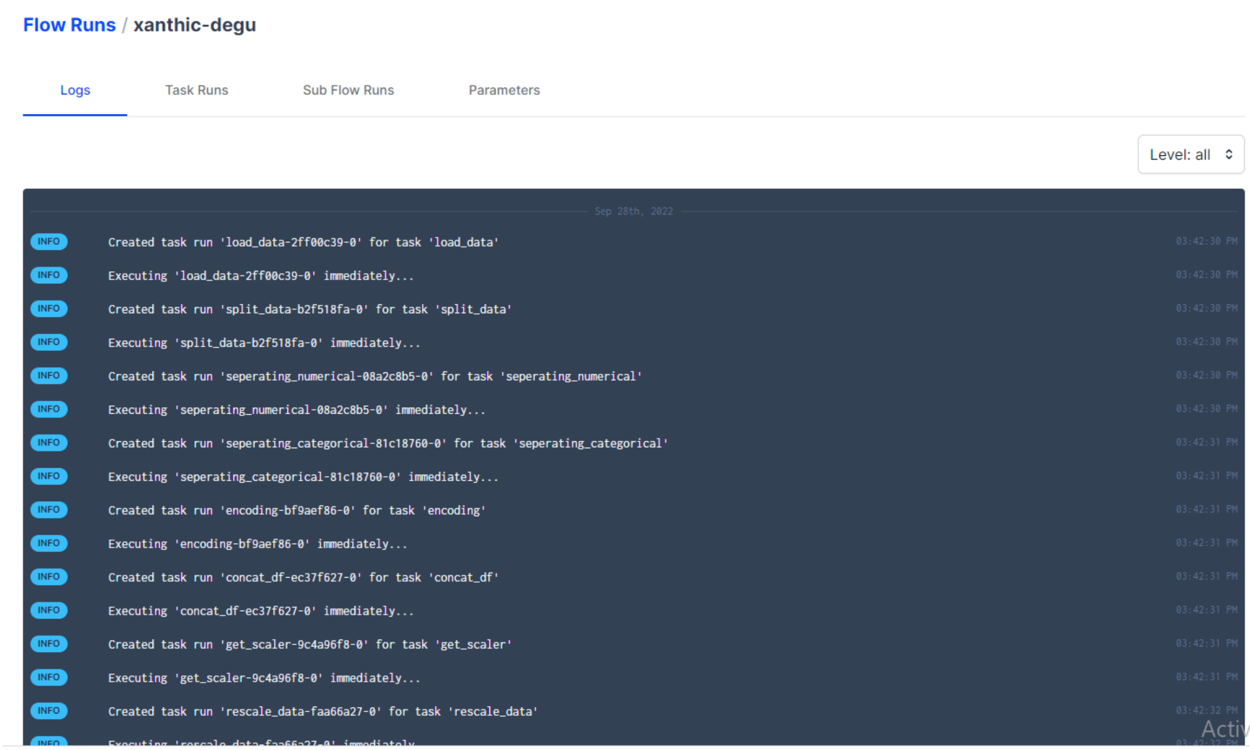




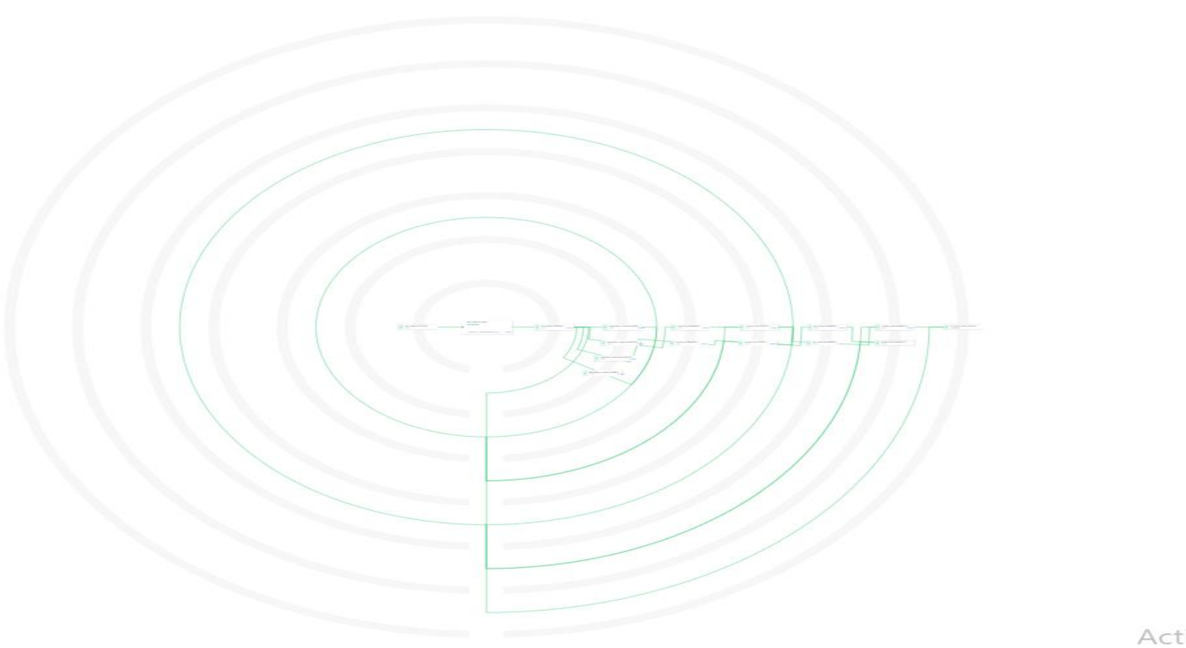
With prefect I applied automatic deployment of model after every 5 minutes interval of time. This features is very helping in industries level project as the real time data is coming and our model updated itself after every 5 minutes to tune itself and give us best possible result.



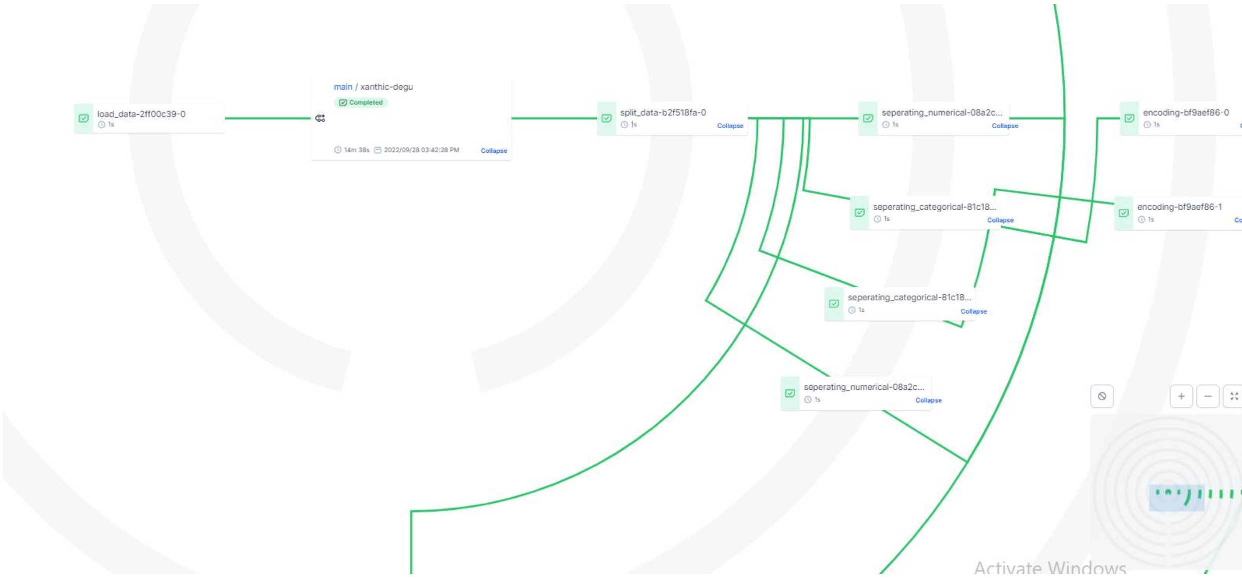
With the prefect we can view all the logs life of our entire machine learning pipeline model

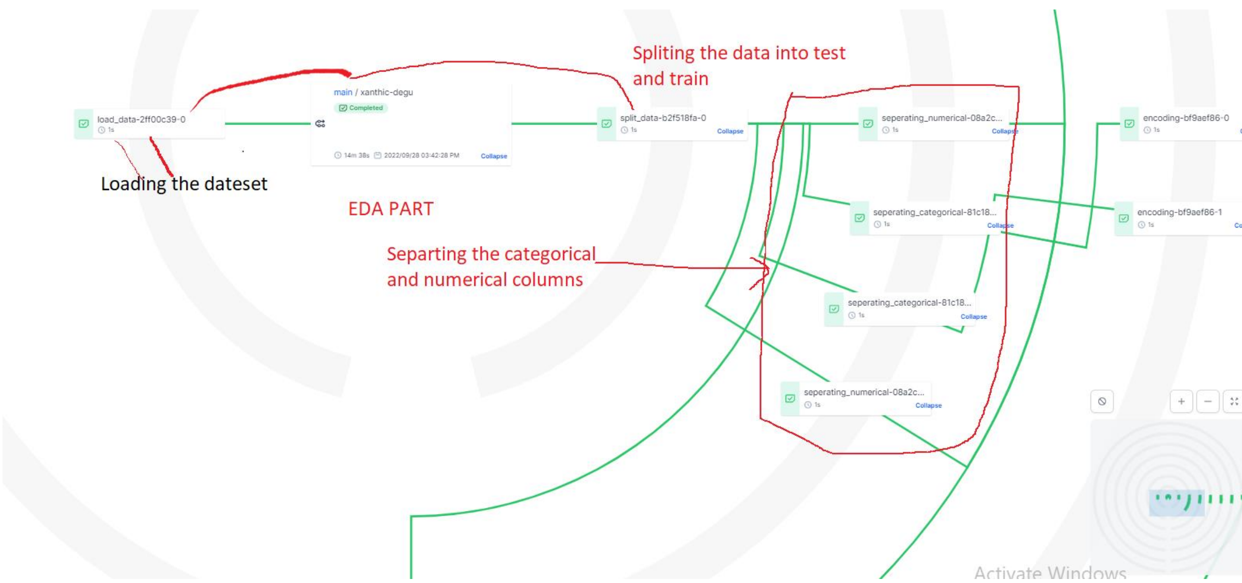


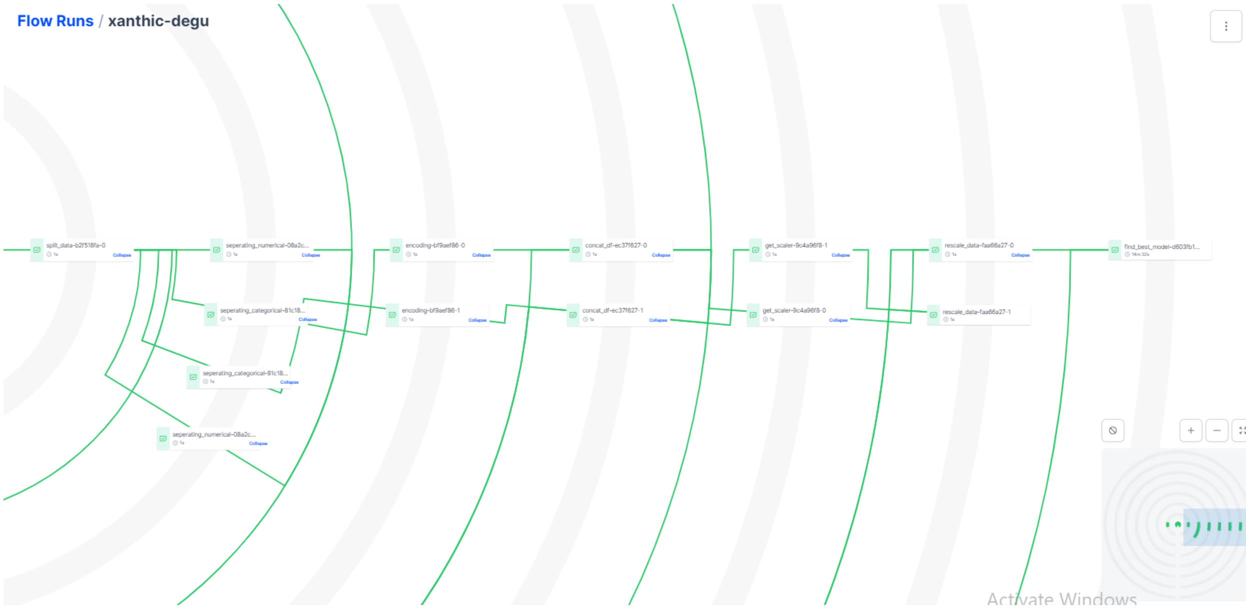
Radar diagram for our pipeline

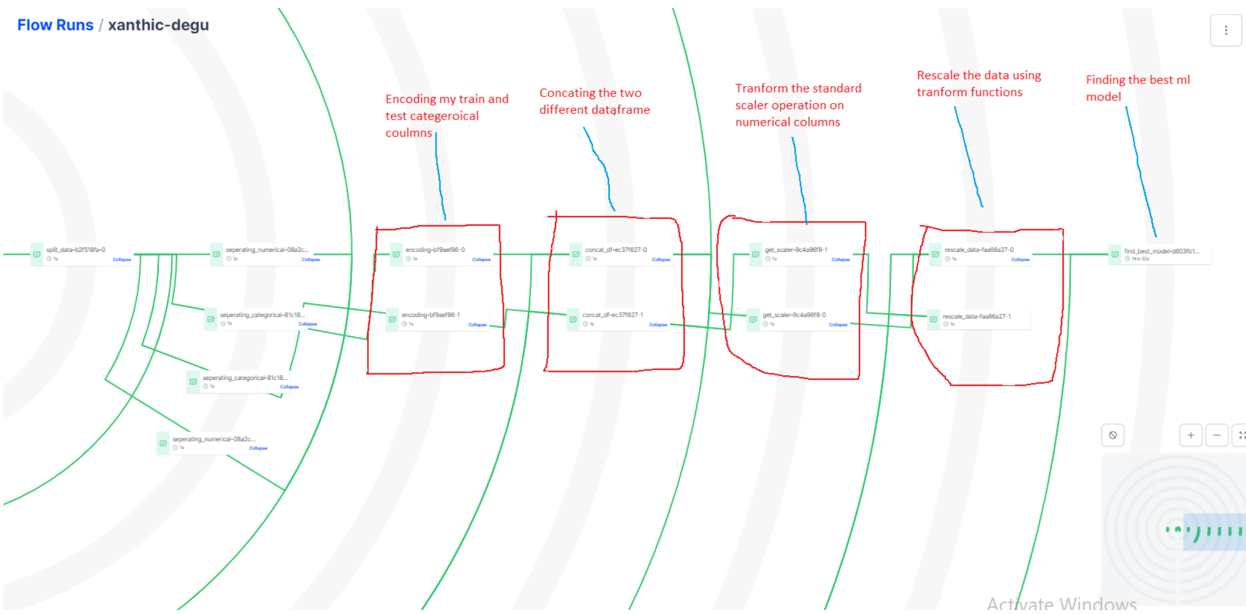


Complete Machine learning model pipeline









Deploying our machine learning model on the cloud

There are two famous cloud for ML model deployment

- Heroku Deployment
- AWS Deployment

Heroku Deployment

Heroku is PaaS Architecture

Along with your application files you need to take care of below mentioned steps:

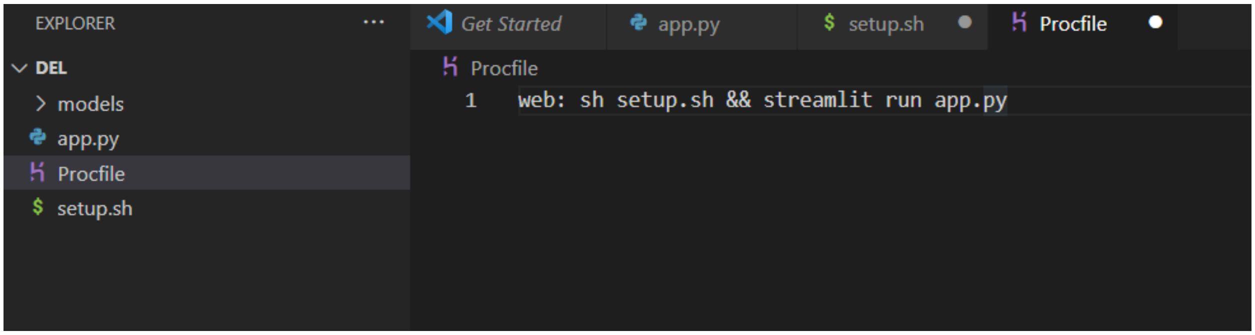
1. You need all the below mentioned files for deployment:
   a. requirements.txt
   b. Procfile
   c. setup.sh
2. Autodetect the app type
3. git push & start running

- Creating setup.sh file

```
mkdir -p ~/.streamlit/
echo "\
[general]\n\
email = \"your@gmail.com\"\n\
" > ~/.streamlit/credentials.toml
echo "\
[server]\n\
headless = true\n\
enableCORS=false\n\
port = $PORT\n\
" > ~/.streamlit/config.toml
```

This is used for configuration of our streamlit application for heroku cloud

- Creating Procfile file



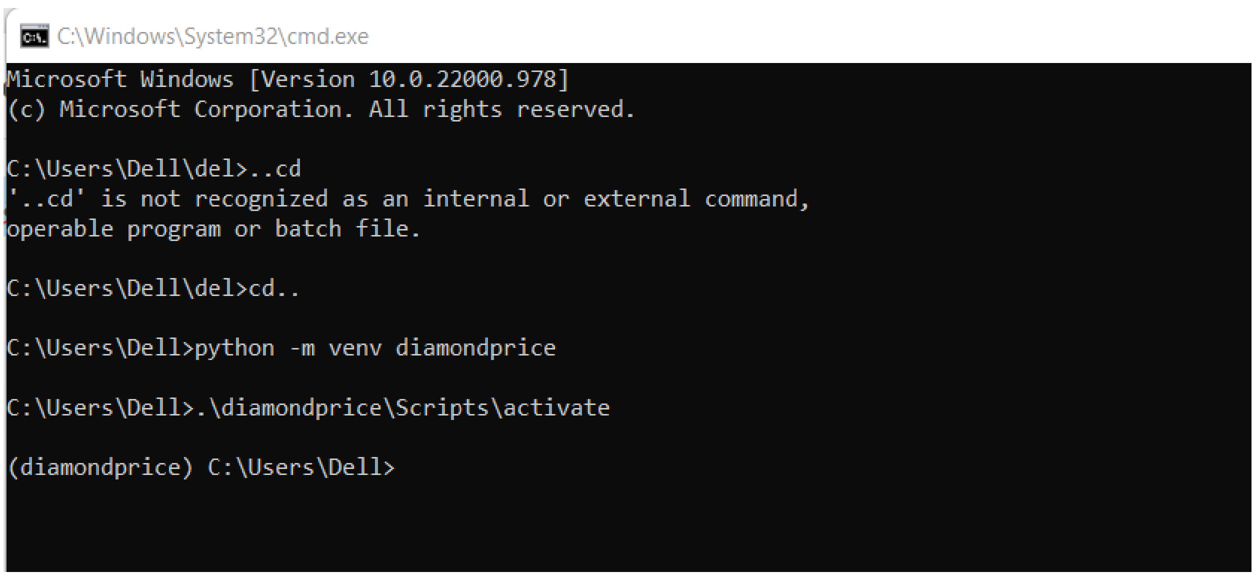
This file is very first command which will run on the cloud to start our application file

- Creating requirements.txt

➢ Create a Python VirtualEnv

`python -m venv diamondprice`

`.\diamondprice\Scripts\activate` => activate virtual environment in Windows



➢ Install these packages in virtualenv:

```
pip install matplotlib seaborn plotly sklearn nltk streamlit
```

```
(diamondprice) C:\Users\Dell>pip install streamlit sklearn
Collecting streamlit
  Using cached streamlit-1.13.0-py2.py3-none-any.whl (9.2 MB)
Collecting sklearn
  Using cached sklearn-0.0.tar.gz (1.1 kB)
  Preparing metadata (setup.py) ... done
Collecting click>=7.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting tornado>=5.0
  Using cached tornado-6.2-cp37-abi3-win_amd64.whl (425 kB)
Collecting pympler>=0.9
  Using cached Pympler-1.0.1-py3-none-any.whl (164 kB)
Collecting numpy
  Using cached numpy-1.23.3-cp310-cp310-win_amd64.whl (14.6 MB)
Collecting rich>=10.11.0
  Using cached rich-12.5.1-py3-none-any.whl (235 kB)
Collecting validators>=0.2
  Using cached validators-0.20.0-py3-none-any.whl
Collecting protobuf!=3.20.2,<4,>=3.12
  Using cached protobuf-3.20.1-cp310-cp310-win_amd64.whl (903 kB)
Collecting toml
  Using cached toml-0.10.2-py2.py3-none-any.whl (16 kB)
Collecting pyarrow>=4.0
  Using cached pyarrow-9.0.0-cp310-cp310-win_amd64.whl (19.5 MB)
Collecting semver
  Using cached semver-2.13.0-py2.py3-none-any.whl (12 kB)
Collecting cachetools>=4.0
  Using cached cachetools-5.2.0-py3-none-any.whl (9.3 kB)
Collecting tzlocal>=1.1
```

- Create requirement.txt file using below mentioned command:

  `pip freeze > requirements.txt`

```
.1

(diamondprice) C:\Users\Dell>cd del

(diamondprice) C:\Users\Dell\del>pip freeze > requirements.txt

(diamondprice) C:\Users\Dell\del>
```

Install Heroku CLI and Git

✓ Now we have to do login setup in our command window with heroku to deploy our model

```
Microsoft Windows [Version 10.0.22000.978]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell\del>heroku
CLI to interact with Heroku

VERSION
  heroku/7.53.0 win32-x64 node-v12.21.0

USAGE
  $ heroku [COMMAND]

COMMANDS
  access          manage user access to apps
  addons          tools and services for developing, extending, and operating your app
  apps            manage apps on Heroku
  auth            check 2fa status
  authorizations  OAuth authorizations
  autocomplete    display autocomplete installation instructions
  buildpacks      scripts used to compile apps
  certs           a topic for the ssl plugin
  ci              run an application test suite on Heroku
  clients         OAuth clients on the platform
  config          environment variables of apps
  container       Use containers to build and deploy Heroku apps
  domains         custom domains for apps
  drains          forward logs to syslog or HTTPS
  features        add/remove app features
  git             manage local git repository for app
  help            display help for heroku
  keys            add/remove account ssh keys
  labs            add/remove experimental features
  local           run Heroku app locally
  logs            display recent log output
  maintenance     enable/disable access to app
```

➢ Create a git repo and commit it locally `git init`

```
git add *
git commit -m "commit_i"
```

C:\Windows\System32\cmd.exe

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Procfile
        app.py
        models/
        requirements.txt
        setup.sh

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\Dell\del>git add .

C:\Users\Dell\del>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Procfile
        new file:   app.py
        new file:   models/knnregression.pkl
        new file:   models/linearregression.pkl
        new file:   models/randomforestregression.pkl
        new file:   models/standard_scaler.pkl
        new file:   requirements.txt
        new file:   setup.sh


C:\Users\Dell\del>git commit -m "commit_i"
[master (root-commit) 02547af] commit_i
 8 files changed, 115 insertions(+)
 create mode 100644 Procfile
 create mode 100644 app.py
 create mode 100644 models/knnregression.pkl
 create mode 100644 models/linearregression.pkl
 create mode 100644 models/randomforestregression.pkl
 create mode 100644 models/standard_scaler.pkl
 create mode 100644 requirements.txt
 create mode 100644 setup.sh
```

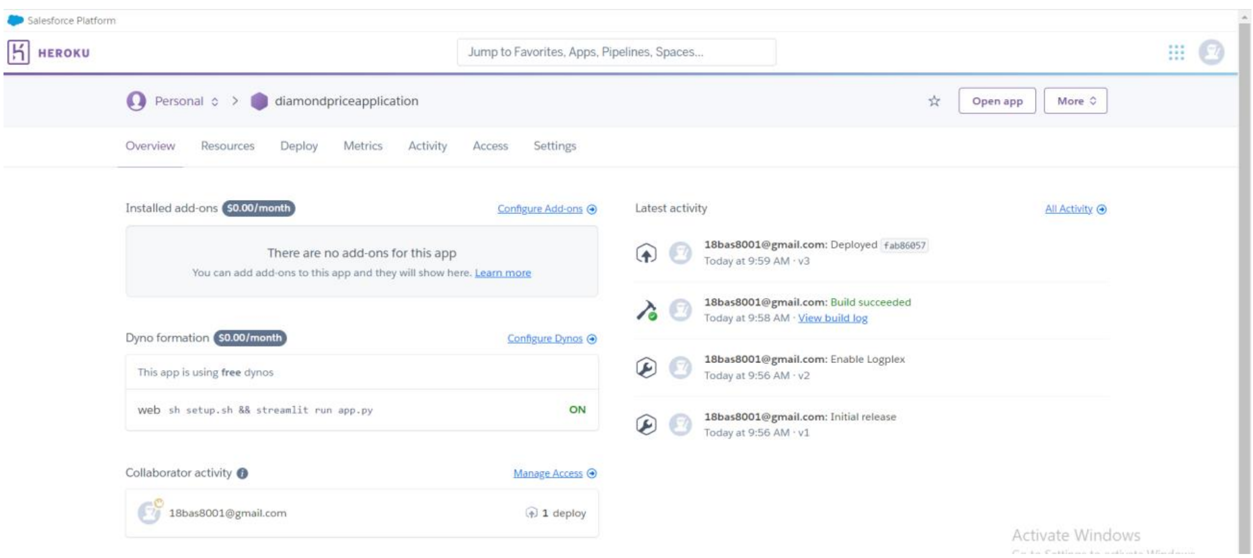
g

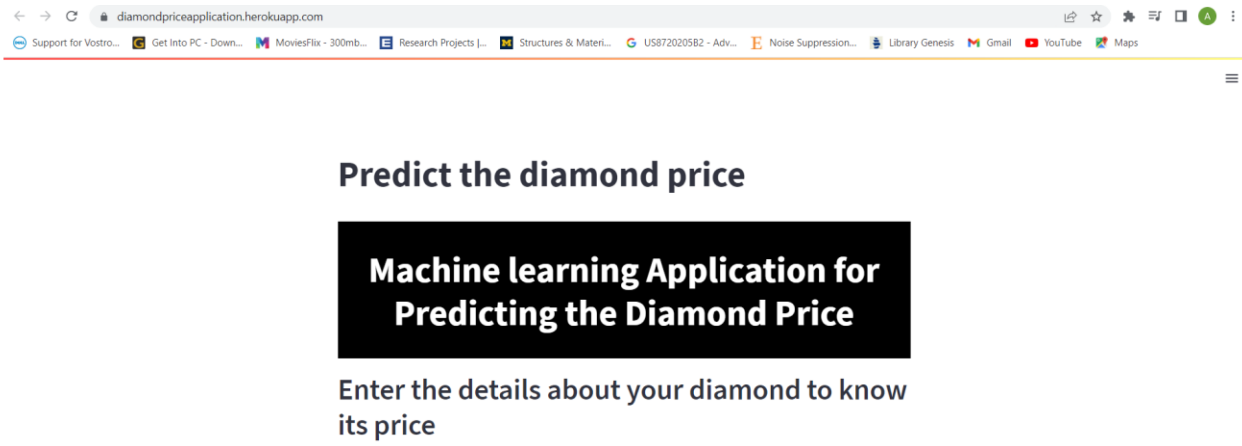
✓ Creating application name and master in our command prompt

```
C:\Users\Dell\del>heroku git:remote -a diamondpricemachinelearning
set git remote heroku to https://git.heroku.com/diamondpricemachinelearning.git

C:\Users\Dell\del>
```

- ✓ git push heroku master

- ✓ Finally we deploy our application on Heroku cloud

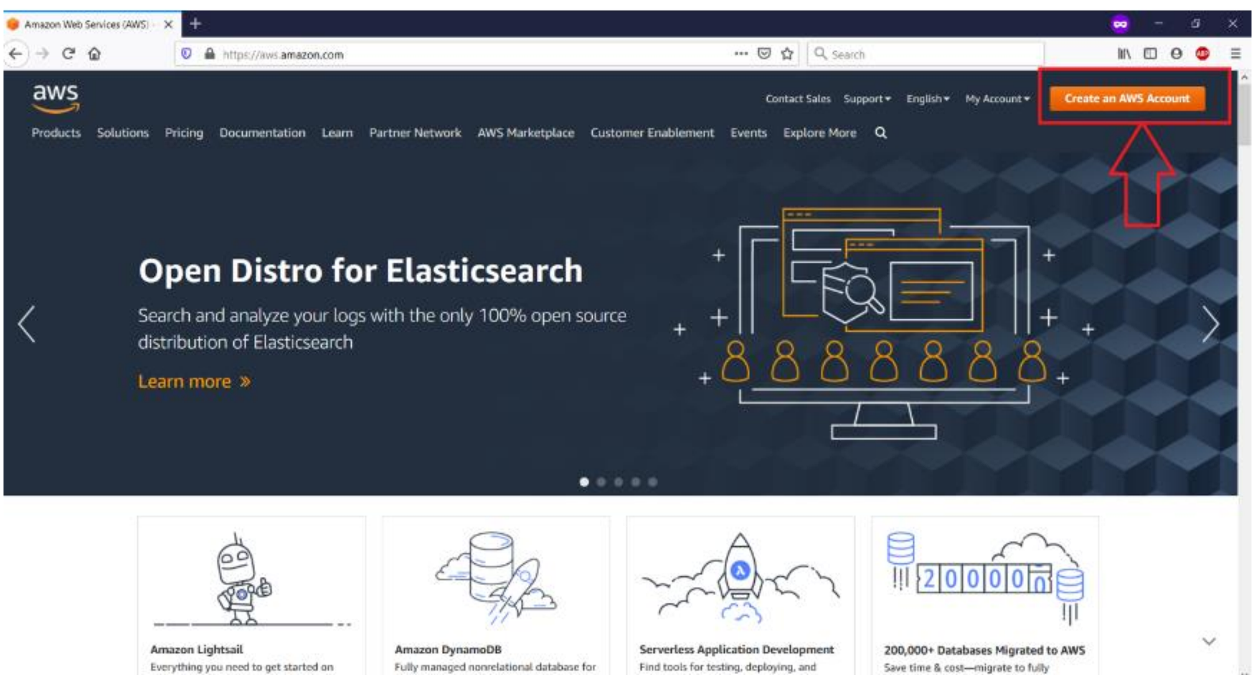




Deployment on AWS

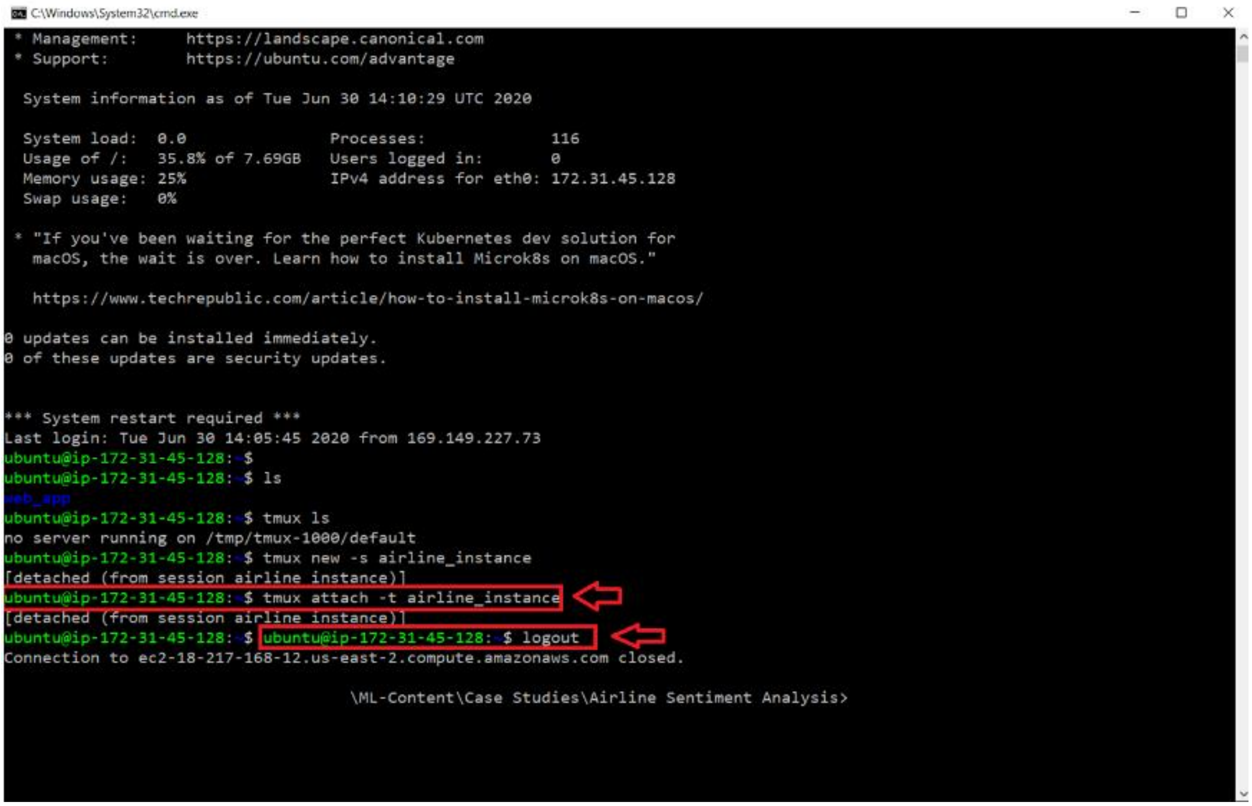
- Step - 1: Create a web application on your computer
- Step - 2: Create AWS Account

Go to aws.amazon.com and click on 'Create an AWS Account'



Step - 3: Create AWS EC2 instance

Step - 4: Hosting the web app on AWS



- Here is your our final application working on AWS cloud