03 mdl

(https://databricks.com)

```
%run ./includes/includes
```

```
Out[3]: DataFrame[]
```

VERY IMPORTANT TO UNDERSTAND THE USE OF THESE VARIABLES!
Please ask if you are confused about their use.

Variable Name	Value	Description
NYC_WEATHER_FILE_PATH	dbfs:/FileStore/tables/raw/weather/	Historic NYC Weather for Model Building
BIKE_TRIP_DATA_PATH	dbfs:/FileStore/tables/raw/bike_trips/	Historic Bike Trip Data for Model Building (Stream this data source)
BRONZE_STATION_INFO_PATH	dbfs:/FileStore/tables/bronze_station_info.delta	Station Information (30 min refresh)
BRONZE_STATION_STATUS_PATH	dbfs:/FileStore/tables/bronze_station_status.delta	Station Status

```
start_date = str(dbutils.widgets.get('01.start_date'))
end_date = str(dbutils.widgets.get('02.end_date'))
hours_to_forecast = int(dbutils.widgets.get('03.hours_to_forecast'))
promote_model = bool(True if
str(dbutils.widgets.get('04.promote_model')).lower() == 'yes' else False)

print(start_date,end_date,hours_to_forecast, promote_model)
print("YOUR CODE HERE...")
```

```
com.databricks.dbutils_v1.InputWidgetNotDefined: No input widget named 01.st
art_date is defined
```

```
import mlflow
import json
import pandas as pd
import numpy as np
from prophet import Prophet, serialize
from prophet.diagnostics import cross_validation, performance_metrics
```

```
from mlflow.tracking.client import MlflowClient

# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(color_codes=True)

# Hyperparameter tuning
import itertools

SOURCE_DATA = f"dbfs:/FileStore/tables/G11/silver/inventory"
ARTIFACT_PATH = GROUP_MODEL_NAME
np.random.seed(12345)

data = (spark.read
    .format("delta")
    .load(SOURCE_DATA))

## Helper routine to extract the parameters that were used to train a specific
instance of the model
def extract_params(pr_model):
    return {attr: getattr(pr_model, attr) for attr in
serialize.SIMPLE_ATTRIBUTES}
```

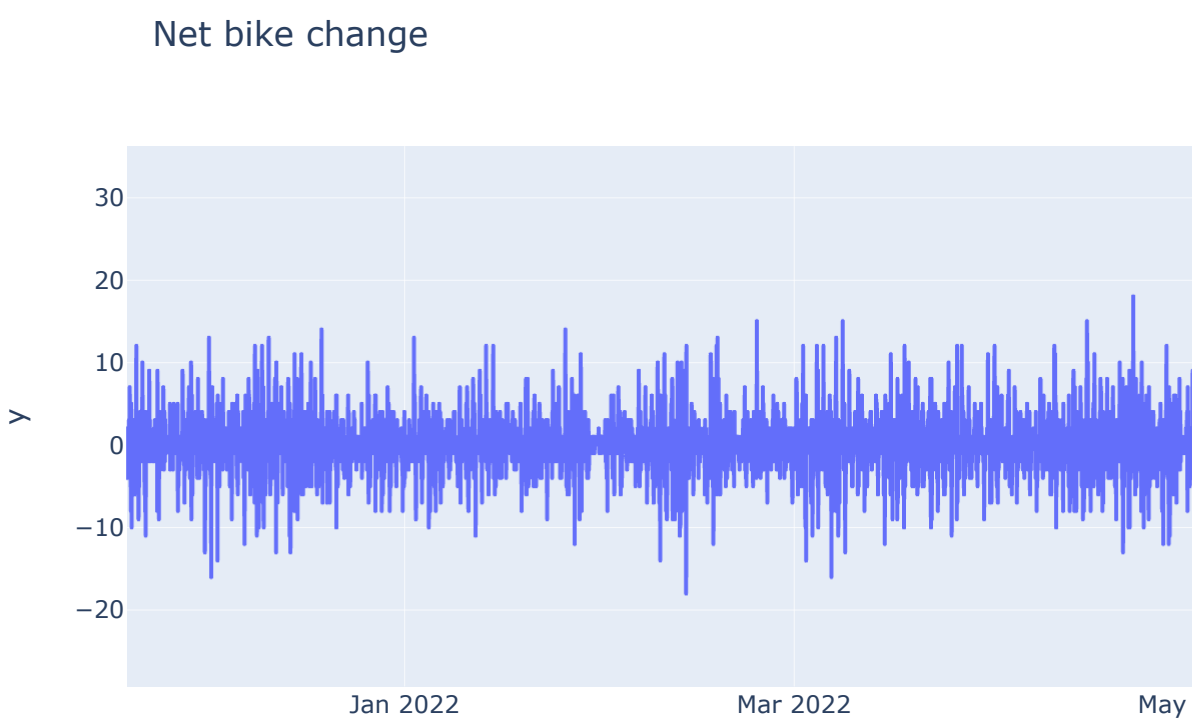
```
df = data.toPandas()
df = df.rename(columns={'dt':'ds', 'net_hour_change':'y'})
print(df)
```

```
                       ds   temp  feels_like  snow_1h    main  rain_1h  y
0      2021-11-19 21:00:00   7.45        3.77      0.0  Clouds      0.0 -3
1      2021-11-19 22:00:00   7.63        4.10      0.0  Clouds      0.0  0
2      2021-11-19 23:00:00   7.61        4.13      0.0  Clouds      0.0 -1
3      2021-11-20 00:00:00   7.35        3.92      0.0  Clouds      0.0 -4
4      2021-11-20 01:00:00   6.82        3.37      0.0  Clouds      0.0  2
...                     ...    ...         ...      ...     ...      ... ..
```

```
12214  2023-04-18 12:00:00   9.09        5.73      0.0  Clouds      0.0  0
12215  2023-04-18 13:00:00   9.74        6.50      0.0  Clouds      0.0  0
12216  2023-04-18 14:00:00  10.60        8.89      0.0  Clouds      0.0  0
12217  2023-04-18 15:00:00  11.32        9.58      0.0  Clouds      0.0  0
12218  2023-04-18 16:00:00  11.62        9.91      0.0  Clouds      0.0  0

[12219 rows x 7 columns]
```

```
#train test split
train_data = df.sample(frac=0.8, random_state=42)
test_data = df.drop(train_data.index)
x_train, y_train, x_test, y_test = train_data["ds"], train_data["y"],
test_data["ds"], test_data["y"]
```

```
import plotly.express as px
fig = px.line(df, x="ds", y="y", title='Net bike change')
fig.show()
```



```
from sklearn.metrics import mean_absolute_error
# Set up parameter grid
param_grid = {
    'changepoint_prior_scale': [0.01, 0.005],
    'seasonality_prior_scale': [4, 8],
    'seasonality_mode': ['additive'],
    'yearly_seasonality' : [True],
    'weekly_seasonality': [True],
    'daily_seasonality': [True]
}

# Generate all combinations of parameters
all_params = [dict(zip(param_grid.keys(), v)) for v in
itertools.product(*param_grid.values())]

print(f"Total training runs {len(all_params)}")

# Create a list to store MAPE values for each combination
maes = []

# Use cross validation to evaluate all parameters
for params in all_params:
    with mlflow.start_run():
        # Fit a model using one parameter combination + holidays
        m = Prophet(**params)
        holidays = pd.DataFrame({"ds": [], "holiday": []})
        m.add_country_holidays(country_name='US')
        m.add_regressor('feels_like')
        m.add_regressor('rain_1h')
        m.add_regressor('temp')
        m.fit(train_data)

        y_pred = m.predict(test_data.dropna())

        mae = mean_absolute_error(y_test.dropna(), y_pred['yhat'])
        mlflow.prophet.log_model(m, artifact_path=ARTIFACT_PATH)
        mlflow.log_params(params)
        mlflow.log_metrics({'mae': mae})
        model_uri = mlflow.get_artifact_uri(ARTIFACT_PATH)
        print(f"Model artifact logged to: {model_uri}")

        # Save model performance metrics for this combination of hyper
parameters
        maes.append((mae, model_uri))
```

```
Total training runs 4
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
Model artifact logged to: dbfs:/databricks/mlflow-tracking/882cc7127c8c4bacb30
954f93f302582/47db7852a21e461ba652f10d70f3f6f6/artifacts/G11_model
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```

```
# Tuning results
tuning_results = pd.DataFrame(all_params)
tuning_results['mae'] = list(zip(*maes))[0]
tuning_results['model']= list(zip(*maes))[1]
best_params =
dict(tuning_results.iloc[tuning_results[['mae']].idxmin().values[0]])
best_params
```

```
INFO:py4j.clientserver:Received command c on object id p1
Out[11]: {'changepoint_prior_scale': 0.005,
 'seasonality_prior_scale': 4,
 'seasonality_mode': 'additive',
 'yearly_seasonality': True,
 'weekly_seasonality': True,
 'daily_seasonality': True,
 'mae': 2.535853674985773,
 'model': 'dbfs:/databricks/mlflow-tracking/882cc7127c8c4bacb30954f93f302582/7
b7ae081e825422fb3eb21c9703cacc3/artifacts/G11_model'}
```

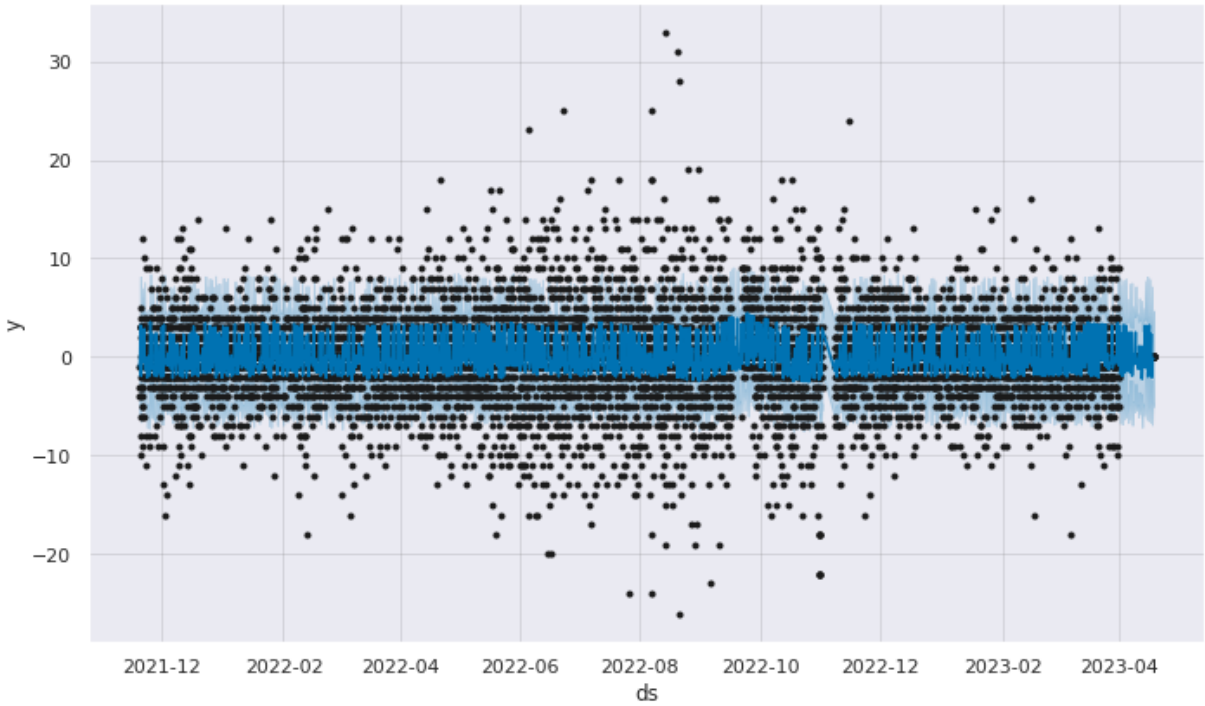
```
loaded_model = mlflow.prophet.load_model(best_params['model'])
forecast = loaded_model.predict(test_data)
print(f"forecast:\n${forecast.tail(40)}")
```

```
INFO:py4j.clientserver:Received command c on object id p1
INFO:py4j.clientserver:Received command c on object id p0
```

```
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
forecast:
$                      ds     trend  ...  multiplicative_terms_upper      yhat
2404 2023-04-11 04:00:00  0.163214  ...                         0.0 -0.259536
2405 2023-04-11 07:00:00  0.163245  ...                         0.0  1.094179
2406 2023-04-11 14:00:00  0.163315  ...                         0.0 -0.337881
2407 2023-04-12 00:00:00  0.163415  ...                         0.0 -1.085152
2408 2023-04-12 12:00:00  0.163536  ...                         0.0  1.094422
2409 2023-04-12 16:00:00  0.163576  ...                         0.0 -1.204764
2410 2023-04-12 18:00:00  0.163596  ...                         0.0 -2.226142
2411 2023-04-13 02:00:00  0.163677  ...                         0.0 -0.464871
```

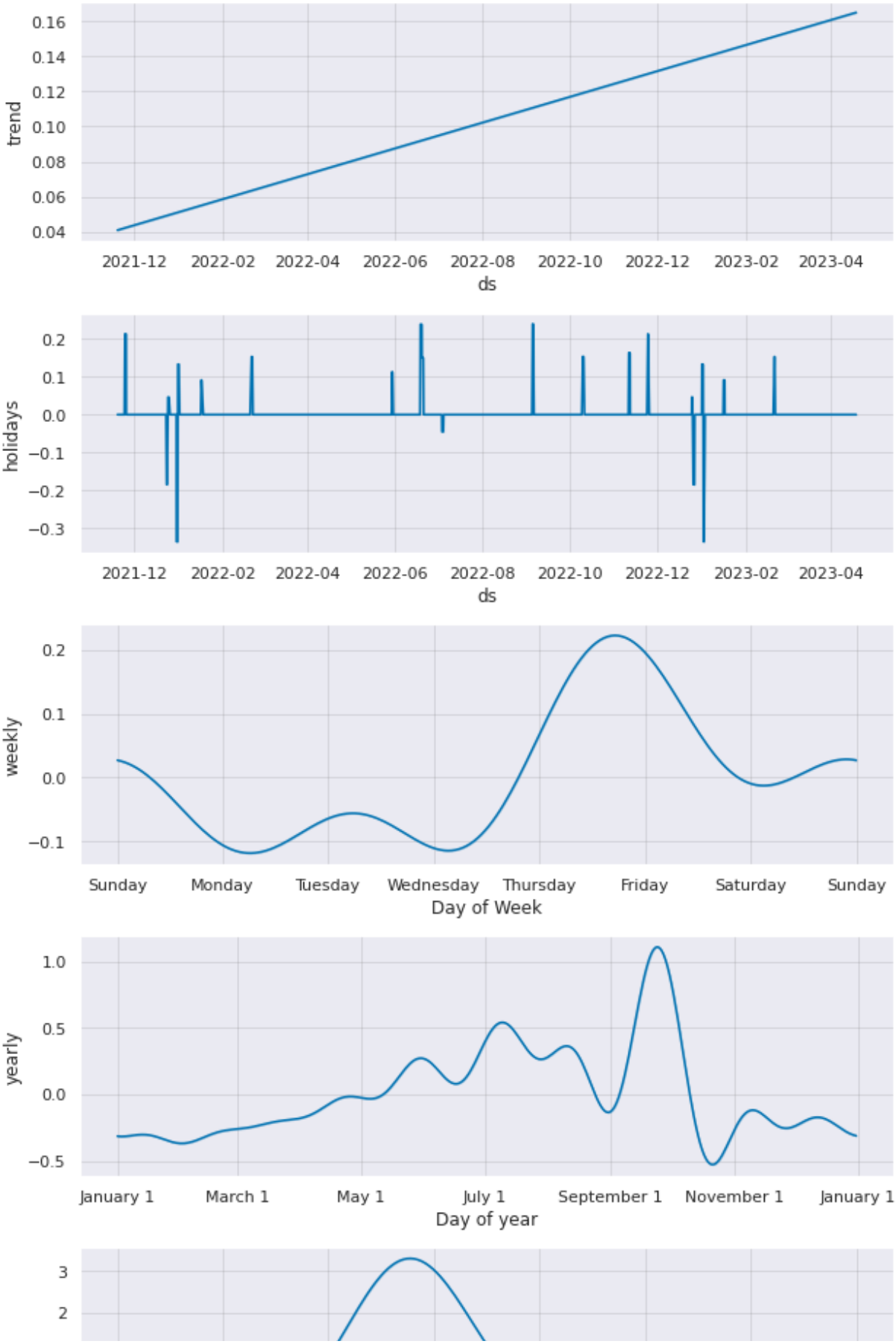
```
prophet_plot = loaded_model.plot(forecast)
```

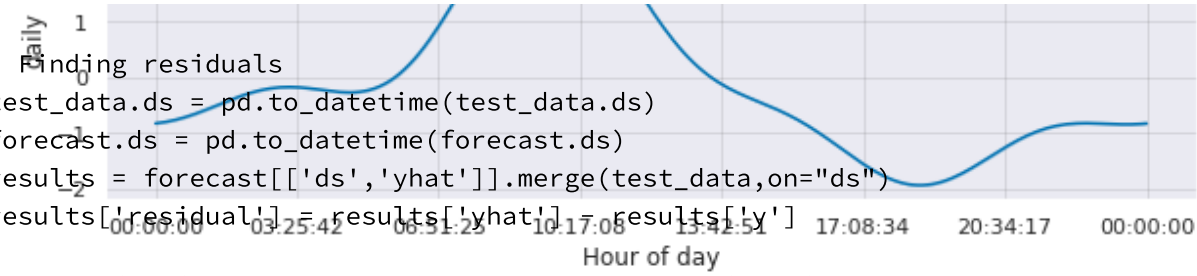
```
INFO:py4j.clientserver:Received command c on object id p1
```




```
prophet_plot2 = loaded_model.plot_components(forecast)
```

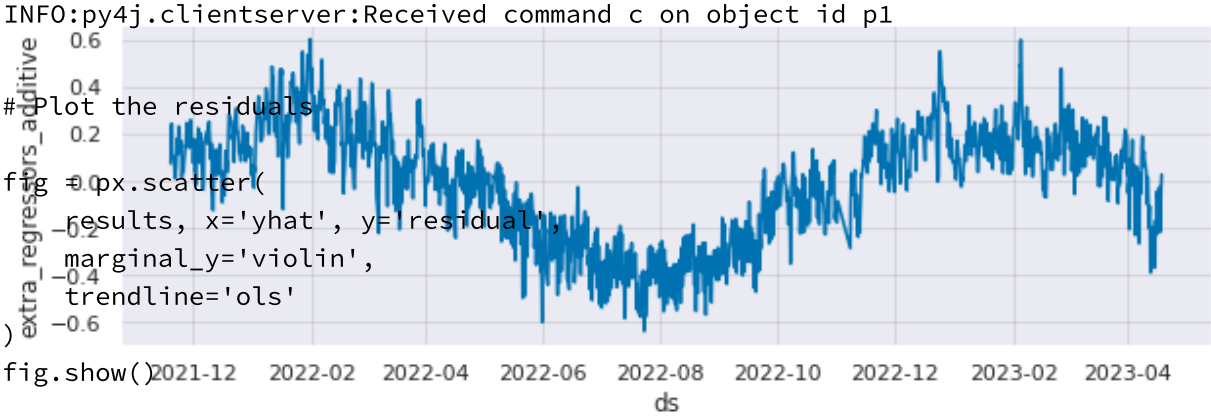
```
INFO:py4j.clientserver:Received command c on object id p1
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```


trend
0.16
0.14
0.12
0.10
0.08
0.06
0.04
2021-12
2022-02
2022-04
2022-06
2022-08
2022-10
2022-12
2023-02
2023-04
ds
holidays
0.2
0.1
0.0
−0.1
−0.2
−0.3
2021-12
2022-02
2022-04
2022-06
2022-08
2022-10
2022-12
2023-02
2023-04
ds
weekly
0.2
0.1
0.0
−0.1
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of Week
yearly
1.0
0.5
0.0
−0.5
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
3
2



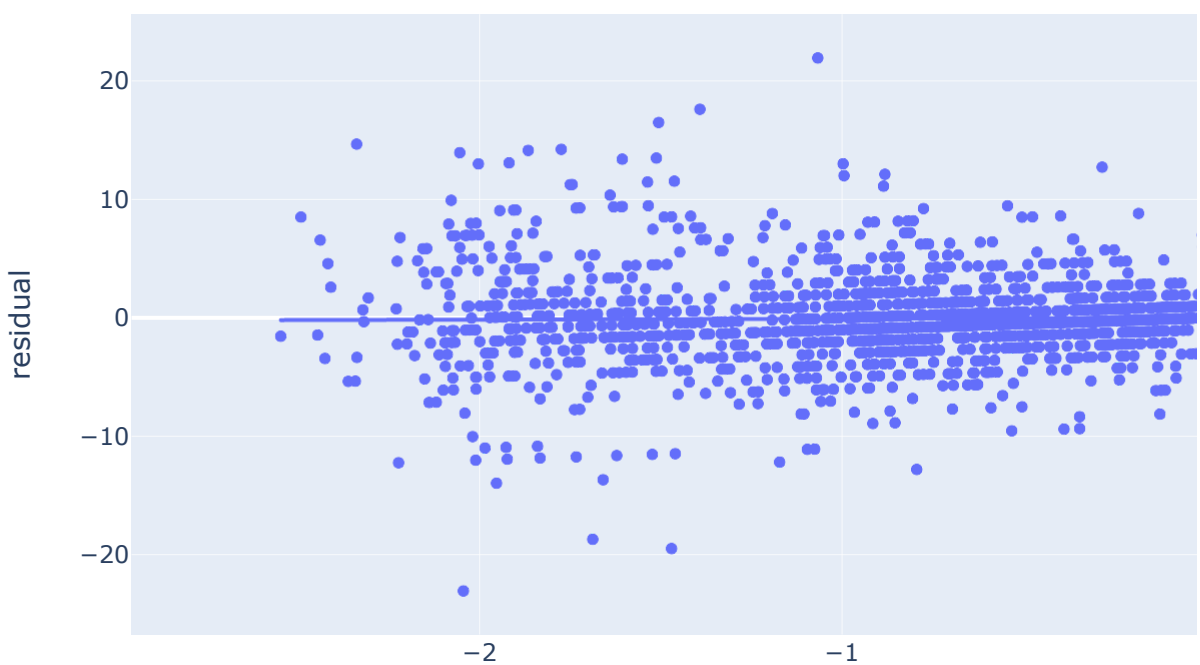

```
# Finding residuals
test_data.ds = pd.to_datetime(test_data.ds)
forecast.ds = pd.to_datetime(forecast.ds)
results = forecast[['ds','yhat']].merge(test_data,on="ds")
results['residual'] = results['yhat'] - results['y']
```

```
INFO:py4j.clientserver:Received command c on object id p1
```




```
# Plot the residuals

fig = px.scatter(
  results, x='yhat', y='residual',
  marginal_y='violin',
  trendline='ols'
)
fig.show()
```

```
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```



```
# Register Model to MLFlow

model_details = mlflow.register_model(model_uri=best_params['model'],
name=GROUP_MODEL_NAME)
```

```
Registered model 'G11_model' already exists. Creating a new version of this mo
del...
2023/05/09 02:26:06 INFO mlflow.tracking._model_registry.client: Waiting up to
300 seconds for model version to finish creation.                    Model na
me: G11_model, version 21
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
Created version '21' of model 'G11_model'.
```

```
client = MlflowClient()
```

```
INFO:py4j.clientserver:Received command c on object id p1
```

```
if promote_model:
    client.transition_model_version_stage(
    name=model_details.name,
    version=model_details.version,
    stage='Production')
else:
    client.transition_model_version_stage(
    name=model_details.name,
    version=model_details.version,
    stage='Staging'
)
```

```
INFO:py4j.clientserver:Received command c on object id p1
INFO:py4j.clientserver:Received command c on object id p0
  NameError: name 'promote_model' is not defined
```

```
model_version_details = client.get_model_version(
    name=model_details.name,
    version=model_details.version
)
print("The current model stage is:
'{stage}'".format(stage=model_version_details.current_stage))
```

```
INFO:py4j.clientserver:Received command c on object id p1
The current model stage is: 'None'
```

```
latest_version_info = client.get_latest_versions(ARTIFACT_PATH, stages=
["Staging"])

latest_staging_version = latest_version_info[0].version

print("The latest staging version of the model '%s' is '%s'." % (ARTIFACT_PATH,
latest_staging_version))
```

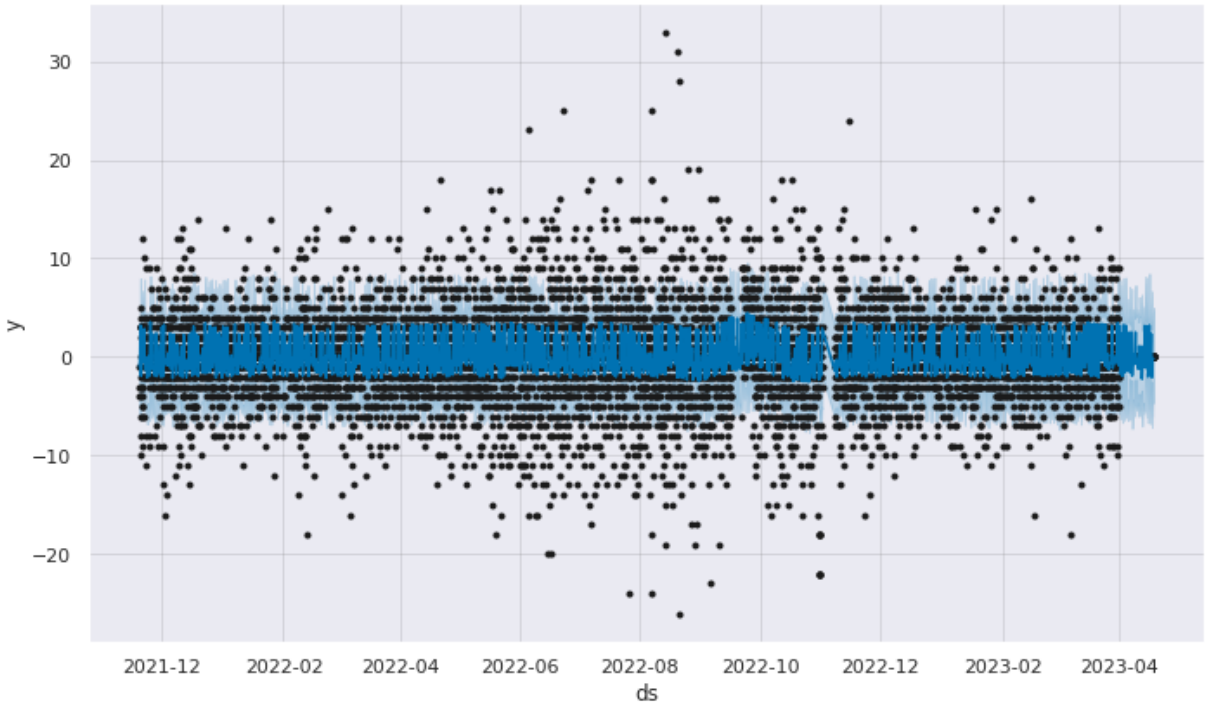
```
INFO:py4j.clientserver:Received command c on object id p1
The latest staging version of the model 'G11_model' is '20'.
```

```
model_staging_uri =
"models:/{model_name}/staging".format(model_name=ARTIFACT_PATH)

print("Loading registered model version from URI:
'{model_uri}'".format(model_uri=model_staging_uri))

model_staging = mlflow.prophet.load_model(model_staging_uri)
```

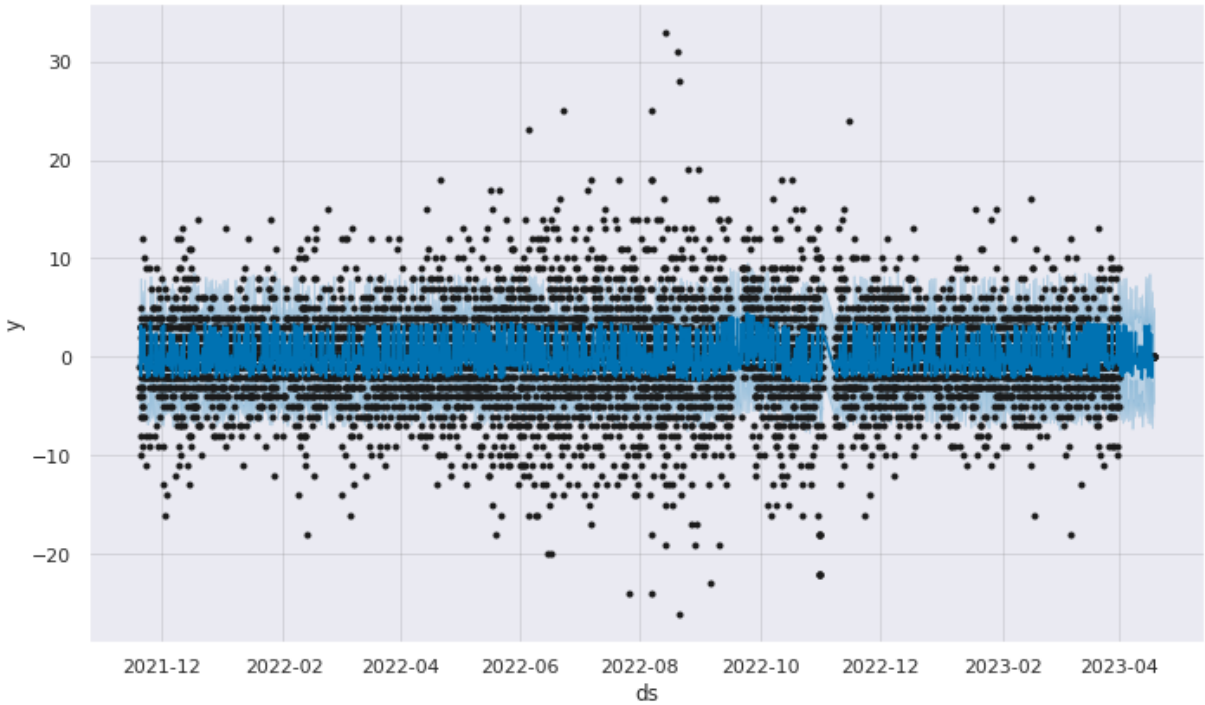
```
INFO:py4j.clientserver:Received command c on object id p1
Loading registered model version from URI: 'models:/G11_model/staging'
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```

```
model_staging.plot(model_staging.predict(test_data))
```

```
INFO:py4j.clientserver:Received command c on object id p1
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```




```
INFO:py4j.clientserver:Received command c on object id p0
```




```
import json

# Return Success
dbutils.notebook.exit(json.dumps({"exit_code": "OK"}))
```

```
Notebook exited: {"exit_code": "OK"}
```