databricks 04 app

(https://databricks.com)

```
pip install folium
```

```
Python interpreter will be restarted.
Collecting folium
  Downloading folium-0.14.0-py2.py3-none-any.whl (102 kB)
Requirement already satisfied: jinja2>=2.9 in /databricks/python3/lib/python3.9/site-packages (from folium) (2.11.3)
Requirement already satisfied: numpy in /databricks/python3/lib/python3.9/site-packages (from folium) (1.20.3)
Collecting branca>=0.6.0
  Downloading branca-0.6.0-py3-none-any.whl (24 kB)
Requirement already satisfied: requests in /databricks/python3/lib/python3.9/site-packages (from folium) (2.26.0)
Requirement already satisfied: MarkupSafe>=0.23 in /databricks/python3/lib/python3.9/site-packages (from jinja2>=2.9->fol
ium) (2.0.1)
Requirement already satisfied: idna<4,>=2.5 in /databricks/python3/lib/python3.9/site-packages (from requests->folium)
(3.2)
Requirement already satisfied: charset-normalizer~=2.0.0 in /databricks/python3/lib/python3.9/site-packages (from request
s->folium) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /databricks/python3/lib/python3.9/site-packages (from requests->f
olium) (1.26.7)
Requirement already satisfied: certifi>=2017.4.17 in /databricks/python3/lib/python3.9/site-packages (from requests->foli
um) (2021.10.8)
Installing collected packages: branca, folium
Successfully installed branca-0.6.0 folium-0.14.0
Python interpreter will be restarted.
```

```
%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 (30 min refresh)
BRONZE_NYC_WEATHER_PATH	dbfs:/FileStore/tables/bronze_nyc_weather.delta	NYC Weather (30 min refresh)
USER_NAME	jtschopp@u.rochester.edu	Email of the user executing this code/notebook
GROUP_NAME	G11	Group Assigment for this user
GROUP_STATION_ASSIGNMENT	Cleveland Pl & Spring St	Station Name to be modeled by this group
GROUP_DATA_PATH	dbfs:/FileStore/tables/G11/	Path to store all of your group data files (delta ect)
GROUP_MODEL_NAME	G11_model	Mlflow Model Name to be used to register your model
GROUP_DB_NAME	G11_db	Group Database to store any managed tables (pre-defined for you)

```
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.start_date is defined
```

```
import plotly.express as px
import pandas as pd
import matplotlib.pyplot as plt
from mlflow.tracking.client import MlflowClient
import datetime
from pyspark.sql.functions import *
import mlflow

ARTIFACT_PATH = GROUP_MODEL_NAME


currentdate = pd.Timestamp.now(tz='US/Eastern').round(freq="H")
fmt = '%Y-%m-%d %H:%M:%S'
currenthour = currentdate.strftime("%Y-%m-%d %H")
currentdate = currentdate.strftime(fmt)
print("The current timestamp is:",currentdate)
```

```
The current timestamp is: 2023-05-08 22:00:00
```

```
client = MlflowClient()
prod_model = client.get_latest_versions(GROUP_MODEL_NAME, stages=['Production'])
stage_model = client.get_latest_versions(GROUP_MODEL_NAME, stages=['Staging'])


print("Production Model Details: ")
print(prod_model)
```

```
Production Model Details:
[<ModelVersion: creation_timestamp=1683385831062, current_stage='Production', description='', last_updated_timestamp=1683
385837539, name='G11_model', run_id='', run_link='', source='dbfs:/databricks/mlflow-tracking/4cf909beec964913819e3fadddb
7a702/695688c4c1ae4e66a64b58e30e9856f0/artifacts/G11_model', status='READY', status_message='', tags={}, user_id='fcolomb
o@u.rochester.edu', version='15'>]
```

```
print("Staging Model Details: ")
print(stage_model)
```

```
Staging Model Details:
[<ModelVersion: creation_timestamp=1683488326629, current_stage='Staging', description='', last_updated_timestamp=1683488
333136, name='G11_model', run_id='', run_link='', source='dbfs:/databricks/mlflow-tracking/92d84d096b6d41749962d38524893b
67/208a0baef3f04e37aafaad33a98750cc/artifacts/G11_model', status='READY', status_message='', tags={}, user_id='rochesterb
igdataclass2021@gmail.com', version='20'>]
```

```
import folium

print("Assigned Station: ", GROUP_STATION_ASSIGNMENT)

# Create a map centered at the given latitude and longitude
lat, lon = 40.722062, -73.997278
map = folium.Map(location=[lat, lon], zoom_start=12)

# Add a marker at the given latitude and longitude
folium.Marker(location=[lat, lon], icon=folium.Icon(color='blue')).add_to(map)

# Display the map
map
```

```
Assigned Station:  Cleveland Pl & Spring St
```

Make this Notebook Trusted to load map: File -> Trust Notebook


```
weather_df = (spark.read
    .format("delta")
    .load('dbfs:/FileStore/tables/G11/silver/weather')
    .toPandas())
print("Current Weather: ")
print(weather_df[weather_df.dt==currentdate].reset_index(drop=True))
```

```
Current Weather:
                   dt   temp  feels_like    main  rain_1h
0  2023-05-08 22:00:00  22.73       22.13  Clouds      0.0
```

```
info = (spark.read
    .format("delta")
    .load('dbfs:/FileStore/tables/G11/silver/station_info'))
print("Station Capacity: ", info.collect()[0][0])
```

```
Station Capacity:  33
```

```
station_status = (spark.read
    .format("delta")
    .load('dbfs:/FileStore/tables/G11/silver/station_status'))
```

```
display(station_status.filter(col("last_reported") <= currenthour).sort(desc("last_reported")).head(1))
now = station_status.filter(col("last_reported") <= currenthour).sort(desc("last_reported"))
bikes_available = now.collect()[0][0]
print(bikes_available)
```

Table

	num_bikes_available	num_bikes_disabled	num_docks_available	last_reported	num_docks_disabled	num_ebikes_
1	14	3	15	2023-05-07 15:04:09	0	6

1 row

```
14
```

```
real_time_inventory = station_status.withColumn("unix_rounded",
(round(unix_timestamp("last_reported")/3600)*3600).cast("timestamp"))
real_time_inventory = real_time_inventory.withColumn("rounded_hour",
date_format(from_unixtime(col("unix_rounded").cast("long")), "yyyy-MM-dd HH:mm:ss"))
real_time_inventory = real_time_inventory.drop("unix_rounded")

from pyspark.sql.functions import col, lag, coalesce
from pyspark.sql.window import Window

w = Window.orderBy("rounded_hour")
real_time_inventory = real_time_inventory.withColumn("diff", col("num_bikes_available") - lag(col("num_bikes_available"),
1).over(w))
real_time_inventory = real_time_inventory.withColumn("diff", coalesce(col("diff"), col("num_bikes_available")))
real_time_inventory = real_time_inventory.orderBy("rounded_hour", ascending=False)
from pyspark.sql.functions import monotonically_increasing_id
real_time_inventory = real_time_inventory.withColumn("index", monotonically_increasing_id())
from pyspark.sql.functions import when
diff = real_time_inventory.withColumn("difference", when(col('index').between(0, 7), None).otherwise(col('diff')))
```

```
weather = (spark.read
    .format("delta")
    .load('dbfs:/FileStore/tables/G11/silver/weather'))
data = weather.join(real_time_inventory, weather.dt == real_time_inventory.rounded_hour, "inner")
test_data = data.toPandas()
test_data = test_data.rename(columns={'dt':'ds'}).rename(columns={'diff': 'y'})
test_data['ds'] = test_data['ds'].apply(pd.to_datetime)
#print(test_data)
```

```
client = MlflowClient()
latest_version_info = client.get_latest_versions(ARTIFACT_PATH, stages=['Production'])
latest_production_version = latest_version_info[0].version
print("The latest production version of the model '%s' is '%s'." %(ARTIFACT_PATH, latest_production_version))

# Predict on the future based on the production model
model_prod_uri = f'models:/{ARTIFACT_PATH}/production'
model_prod = mlflow.prophet.load_model(model_prod_uri)
prod_forecast = model_prod.predict(test_data)
prod_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

```
The latest production version of the model 'G11_model' is '15'.
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
```

	ds	yhat	yhat_lower	yhat_upper
0	2023-03-23 00:00:00	-0.773360	-5.617230	4.342740
1	2023-03-23 00:00:00	-0.773360	-5.414173	3.974198
2	2023-03-23 00:00:00	-0.773360	-5.457740	4.105931
3	2023-03-23 01:00:00	-0.566953	-5.282526	4.304785
4	2023-03-23 01:00:00	-0.566953	-5.593271	3.908248
...	...	...	...	...
2158	2023-05-07 13:00:00	0.288017	-4.627170	5.109529
2159	2023-05-07 14:00:00	-0.295111	-5.179548	4.609346
2160	2023-05-07 14:00:00	-0.295111	-5.346982	4.464474
2161	2023-05-07 15:00:00	-0.656942	-5.411053	4.422809
2162	2023-05-07 15:00:00	-0.656942	-5.444142	4.233260

2163 rows × 4 columns

```
display(test_data)
```

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

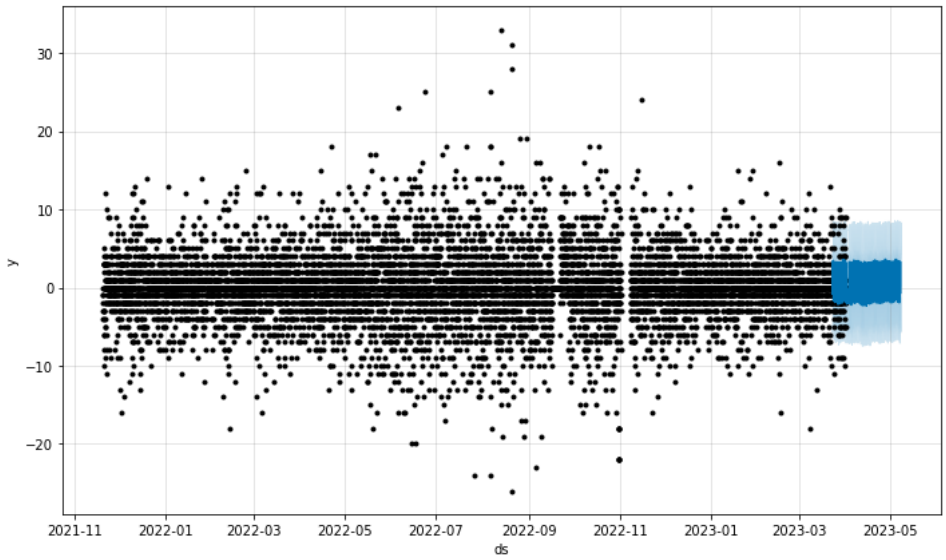
Table

	ds	temp	feels_like	main	rain_1h	num_bikes_available	num_bikes_disab
1	2023-05-07T15:00:00.000+0000	18.29	17.22	Clouds	0	7	3
2	2023-05-07T15:00:00.000+0000	18.29	17.22	Clouds	0	14	3
3	2023-05-07T14:00:00.000+0000	17.15	16.12	Clouds	0	10	3
4	2023-05-07T14:00:00.000+0000	17.15	16.12	Clouds	0	10	3
5	2023-05-07T13:00:00.000+0000	15.4	14.33	Clouds	0	9	3
6	2023-05-07T13:00:00.000+0000	15.4	14.33	Clouds	0	9	3
7	2023-05-07T12:00:00.000+0000	13.59	12.49	Clouds	0	9	3

2,163 rows

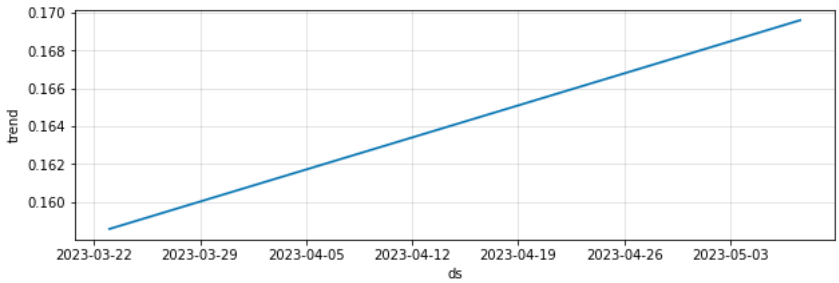
```
prophet_plot = model_prod.plot(prod_forecast)
```

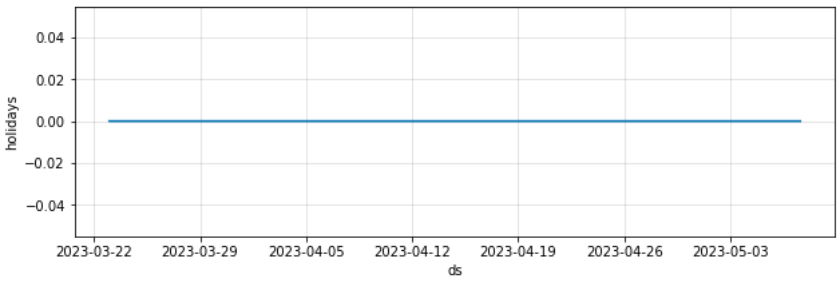
```
INFO:py4j.clientserver:Received command c on object id p1
```

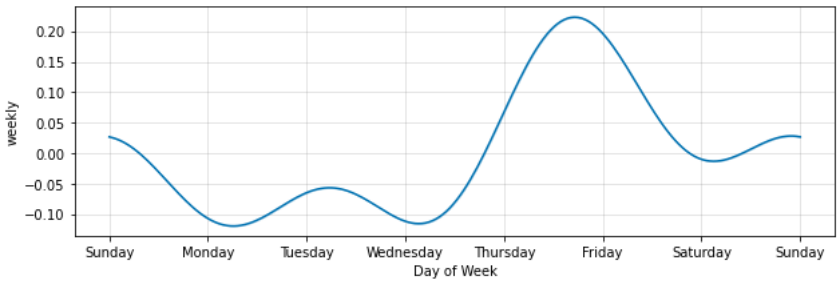


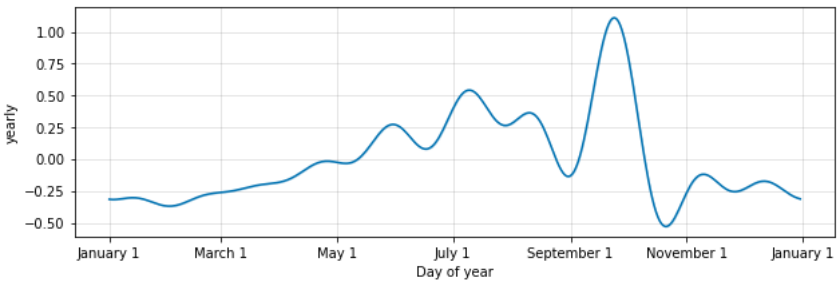

```
prophet_plot2 = model_prod.plot_components(prod_forecast)
```

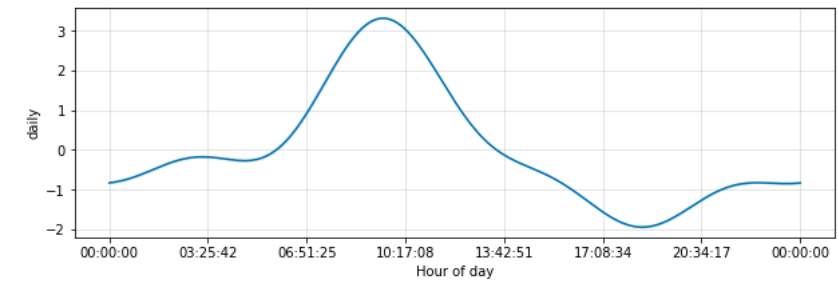
```
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```

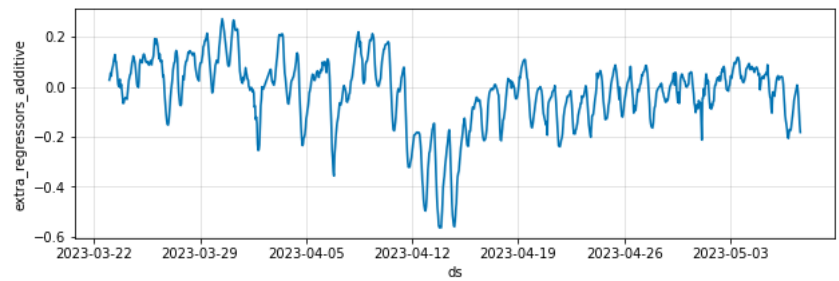

trend
0.170
0.168
0.166
0.164
0.162
0.160
2023-03-22
2023-03-29
2023-04-05
2023-04-12
2023-04-19
2023-04-26
2023-05-03
ds


holidays
0.04
0.02
0.00
−0.02
−0.04
2023-03-22
2023-03-29
2023-04-05
2023-04-12
2023-04-19
2023-04-26
2023-05-03
ds


weekly
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of Week


yearly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year


daily
3
2
1
0
−1
−2
00:00:00
03:25:42
06:51:25
10:17:08
13:42:51
17:08:34
20:34:17
00:00:00
Hour of day


extra_regressors_additive
0.2
0.0
−0.2
−0.4
−0.6
2023-03-22
2023-03-29
2023-04-05
2023-04-12
2023-04-19
2023-04-26
2023-05-03
ds

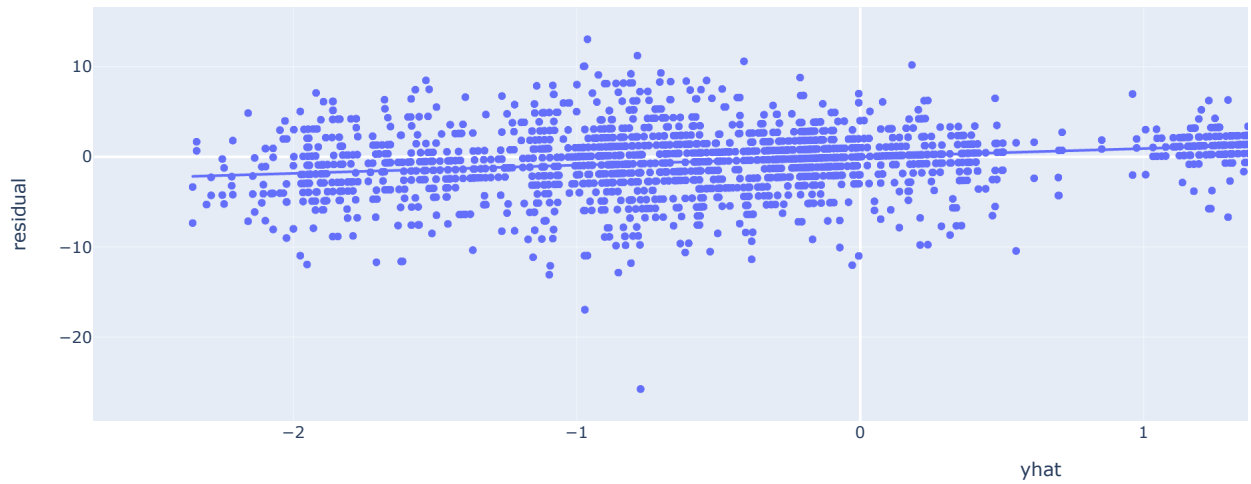
Residual of production model

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

# 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 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
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
```



Gold table for results from production model

```
production_results_path = f"dbfs:/FileStore/tables/G11/gold/production_results"
results_df = spark.createDataFrame(results)
results_production = (results_df.write
    .format("delta")
    .mode("overwrite")
    .save(production_results_path))
```

```
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
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
```

```
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))

# Predict on the future based on the staging model
model_staging_uri = f'models:/{ARTIFACT_PATH}/staging'
model_staging = mlflow.prophet.load_model(model_staging_uri)
staging_forecast = model_staging.predict(test_data)
staging_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

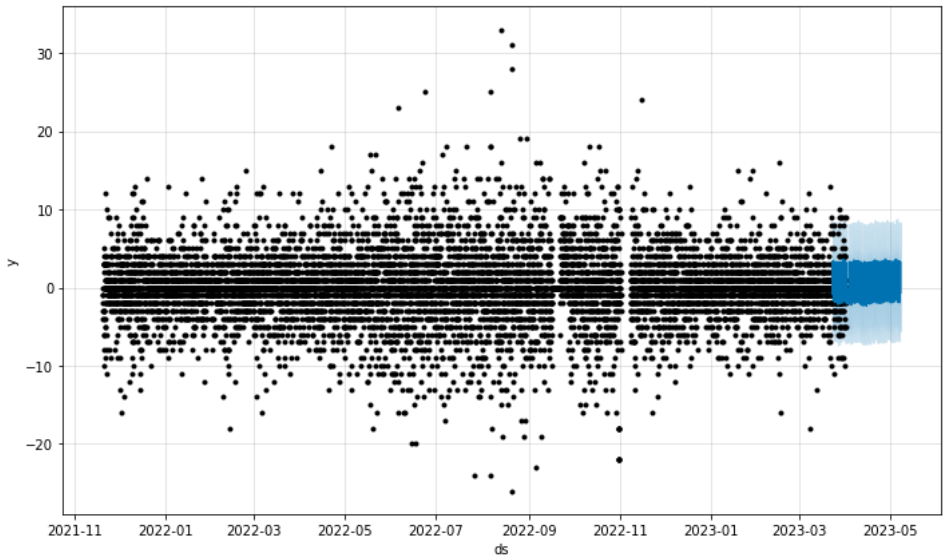
```
INFO:py4j.clientserver:Received command c on object id p1
The latest staging version of the model 'G11_model' is '20'.
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received 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
```

	ds	yhat	yhat_lower	yhat_upper
0	2023-03-23 00:00:00	-0.773360	-5.769119	4.172781
1	2023-03-23 00:00:00	-0.773360	-5.547155	4.232051
2	2023-03-23 00:00:00	-0.773360	-5.318682	4.026777
3	2023-03-23 01:00:00	-0.566953	-5.231182	4.673692
4	2023-03-23 01:00:00	-0.566953	-5.208155	4.519621
...	...	...	...	...
2158	2023-05-07 13:00:00	0.288017	-4.585218	5.142186
2159	2023-05-07 14:00:00	-0.295111	-5.182293	4.161974
2160	2023-05-07 14:00:00	-0.295111	-4.926884	4.838628
2161	2023-05-07 15:00:00	-0.656942	-5.562867	4.211851
2162	2023-05-07 15:00:00	-0.656942	-5.290341	4.218118

2163 rows × 4 columns

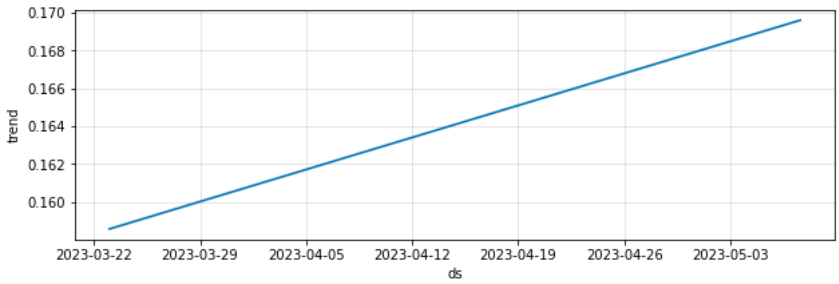
```
prophet_plot = model_staging.plot(staging_forecast)
```

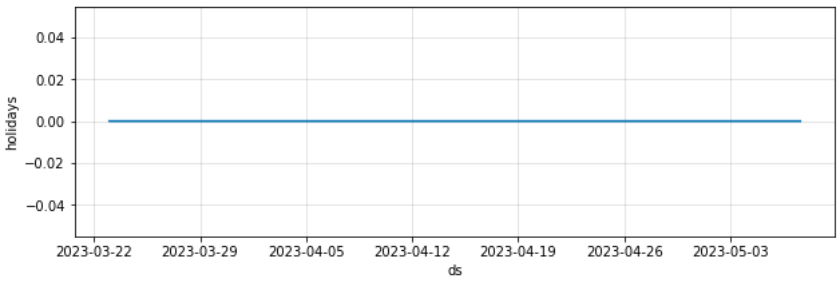
```
INFO:py4j.clientserver:Received command c on object id p1
INFO:py4j.clientserver:Received command c on object id p0
```

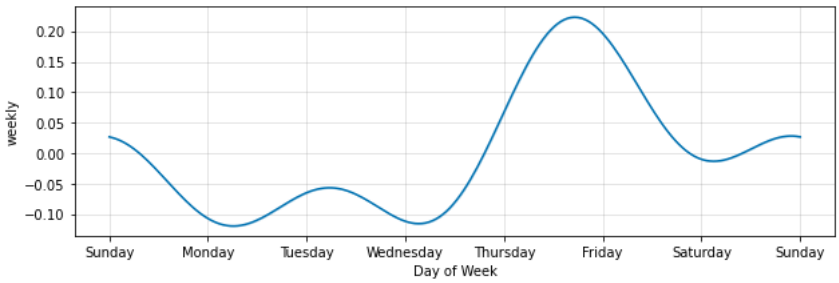


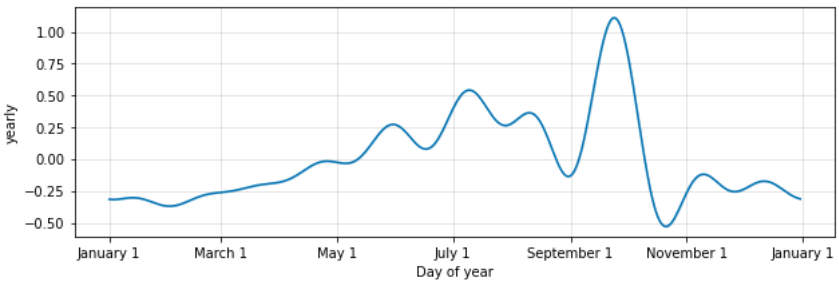

```
prophet_plot2 = model_staging.plot_components(staging_forecast)
```

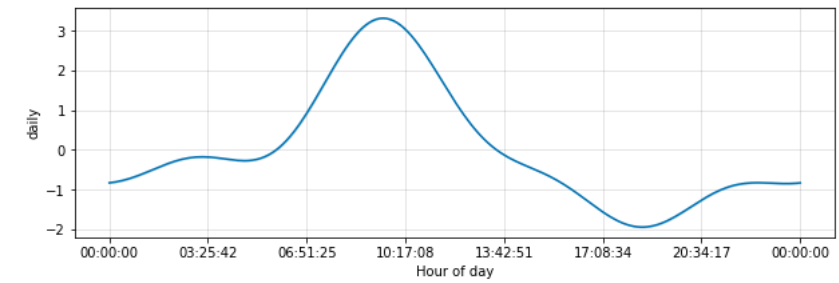
```
INFO:py4j.clientserver:Received command c on object id p1
INFO:py4j.clientserver:Received command c on object id p0
INFO:py4j.clientserver:Received command c on object id p0
```

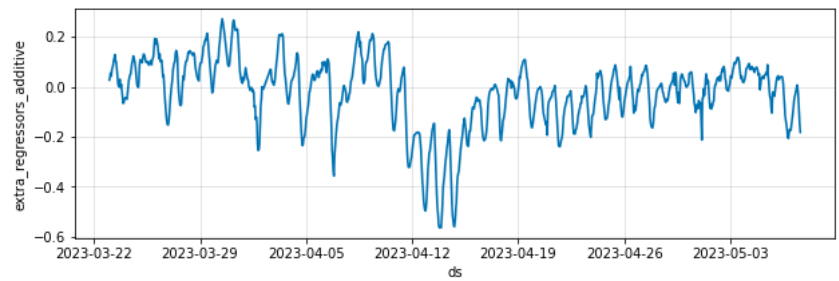

trend
0.170
0.168
0.166
0.164
0.162
0.160
2023-03-22
2023-03-29
2023-04-05
2023-04-12
2023-04-19
2023-04-26
2023-05-03
ds


holidays
0.04
0.02
0.00
−0.02
−0.04
2023-03-22
2023-03-29
2023-04-05
2023-04-12
2023-04-19
2023-04-26
2023-05-03
ds


weekly
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of Week


yearly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year


daily
3
2
1
0
−1
−2
00:00:00
03:25:42
06:51:25
10:17:08
13:42:51
17:08:34
20:34:17
00:00:00
Hour of day


extra_regressors_additive
0.2
0.0
−0.2
−0.4
−0.6
2023-03-22
2023-03-29
2023-04-05
2023-04-12
2023-04-19
2023-04-26
2023-05-03
ds

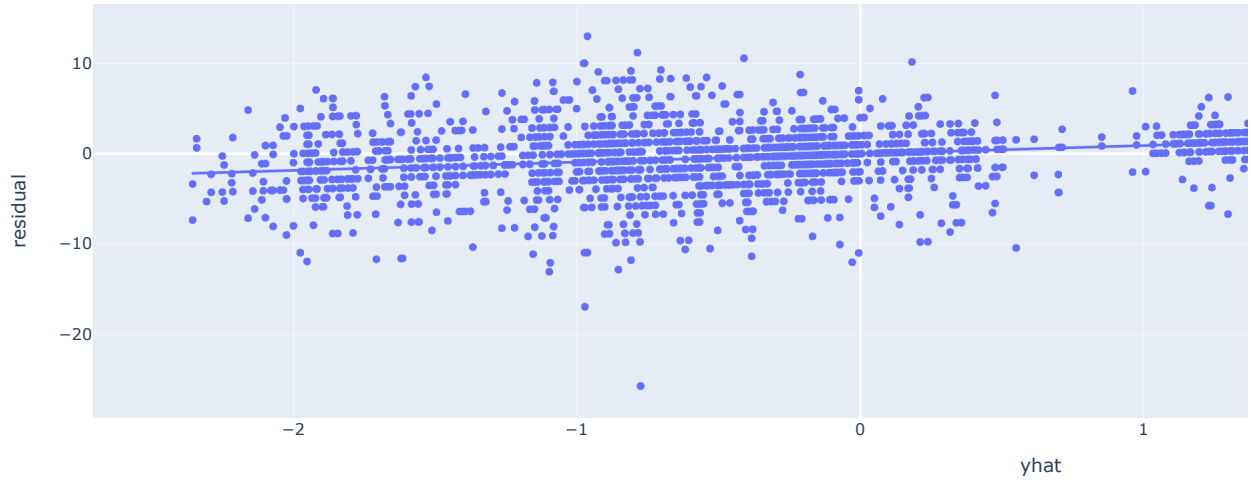
Residual of Staging Model

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

# 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 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
```



Gold table for output of staging model

```
staging_results_path = f"dbfs:/FileStore/tables/G11/gold/staging_results"
results_df = spark.createDataFrame(results)
results_staging = (results_df.write
    .format("delta")
    .mode("overwrite")
    .save(staging_results_path))
```

```
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
```

Here it is only forecasting the next 4 hours

```
#I managed to get it to forecast only the hours specified in the widget but can't figure out how to show that more in
detail
from datetime import timedelta
currentdate = pd.to_datetime(currentdate)
end_pred = currentdate + timedelta(hours=hours_to_forecast)
future_dates = pd.DataFrame(pd.date_range(start=currentdate, end= end_pred, freq='H'), columns=['ds'])
future_df = spark.createDataFrame(future_dates)
df = weather.join(future_df, future_df.ds == weather.dt, 'inner')
test = df.toPandas()
#df.display()
forecast = model_prod.predict(test)
forecast['yhat']= forecast['yhat'] + bikes_available
forecast['yhat_lower']= forecast['yhat_lower'] + bikes_available
forecast['yhat_upper']= forecast['yhat_upper'] + bikes_available
print(forecast)
fig = model_prod.plot(forecast)
ax = fig.gca()
ax.set_xlim(currentdate, end_pred)
ax.set_ylim(-5, 35)
ax.axhline(y=33, color='red')
ax.axhline(y=0, color='red')
```

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

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