Stock Predictions with Prophet 💰

Now that you have seen how `prophet` helps us do predictions, it's **your turn** to do **stock predictions**.

In this challenge, we're looking at the stock closing prices of **Apple** (`AAPL`) starting from 2018-2021. We're using data from a CSV file, so you can plug-in any stock data you want in the future. The data was gathered from IEX.

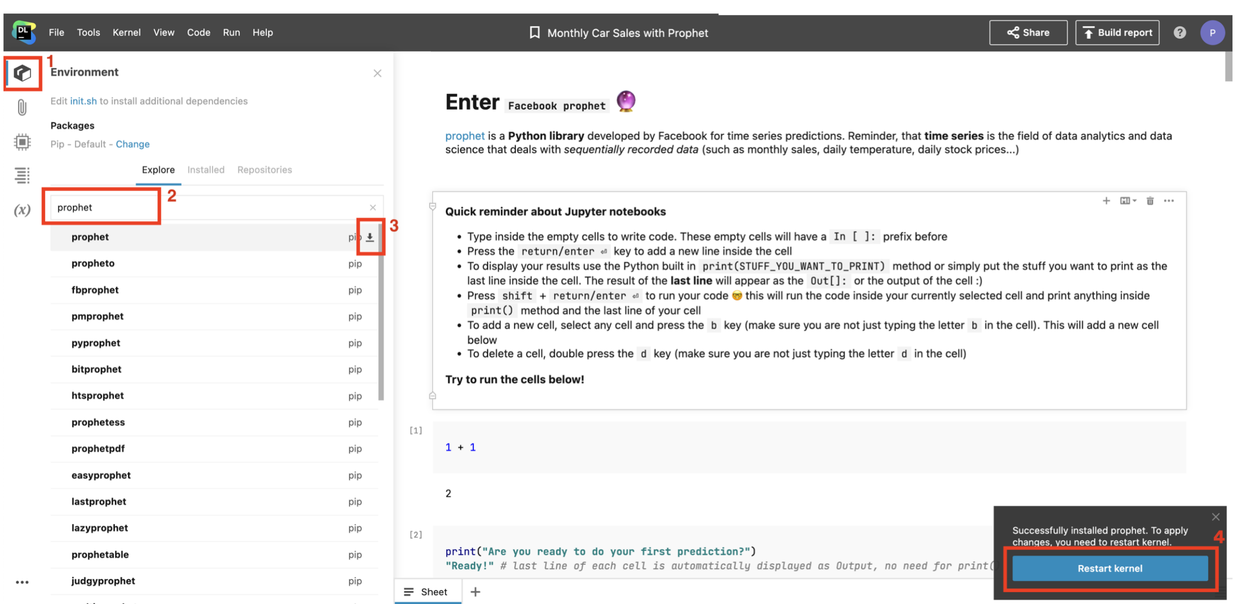
Let's get started 💹

Setup ⚙️

Before working on the code, let's **install `prophet` in this notebook**.

The installation will **take a few minutes** so feel free to start the installation first, so you're ready to go when the exercises start 🚀

ℹ️ Following the image below, click on the "Environment" tab (box icon) and in the "Explore" tab search for `prophet` - install the first package on the list and then click on the "Restart Kernel" button when it appears on the bottom right.



First, we import the necessary Python libraries. `pandas` for dealing with CSV files and data, and `prophet` for our predictions:

```
import pandas as pd
import numpy as np
from prophet import Prophet
```

The rest of the steps are essentially the same as with the car sales forecasting we just did 🚗

So make sure to check back in the `Monthly Car Sales with Prophet` notebook or the lecture slides on Learn if you forgot anything. We hide the solutions for this one, because we know you got this 💪

Your turn! 🚀

Create a DataFrame called `df` by reading the `aapl.csv` file, which is in the `data` folder

```
# your code here
df = pd.read_csv('https://wagon-public-datasets.s3.amazonaws.com/sprints/prophet-aapl-stoc
df
```

	close	date
0	53.060	2018-10-29
1	53.325	2018-10-30
2	54.715	2018-10-31
3	55.555	2018-11-01
4	51.870	2018-11-02
...	...	...
681	148.480	2021-07-15
682	146.390	2021-07-16
683	142.450	2021-07-19
684	146.150	2021-07-20
685	145.400	2021-07-21

686 rows × 2 columns

▶ Solution

Check how many **rows and columns** do you have. Also check what are the **data types** of your columns

```
# number of rows and columns
df.shape
```

```
(686, 2)
```

```
# data types
df.dtypes
```

```
close    float64
date      object
dtype: object
```

▶ Solution

Preparing data for `prophet`

In the livecode we saw that `prophet` asks us to format the data in a certain way to make it work.

Change the columns to `y` and `ds`, in this order. `y` is our stock price (our target to predict), `ds` is the date.

```
# your code here
df.columns = ['y','ds']
```

▶ Solution

Convert the `ds` column to a `datetime` data type. Remember the pandas.to_datetime() function

```
# your code here
df['ds'] = pd.to_datetime(df['ds'])
df.dtypes
```

```
y              float64
ds      datetime64[ns]
dtype: object
```

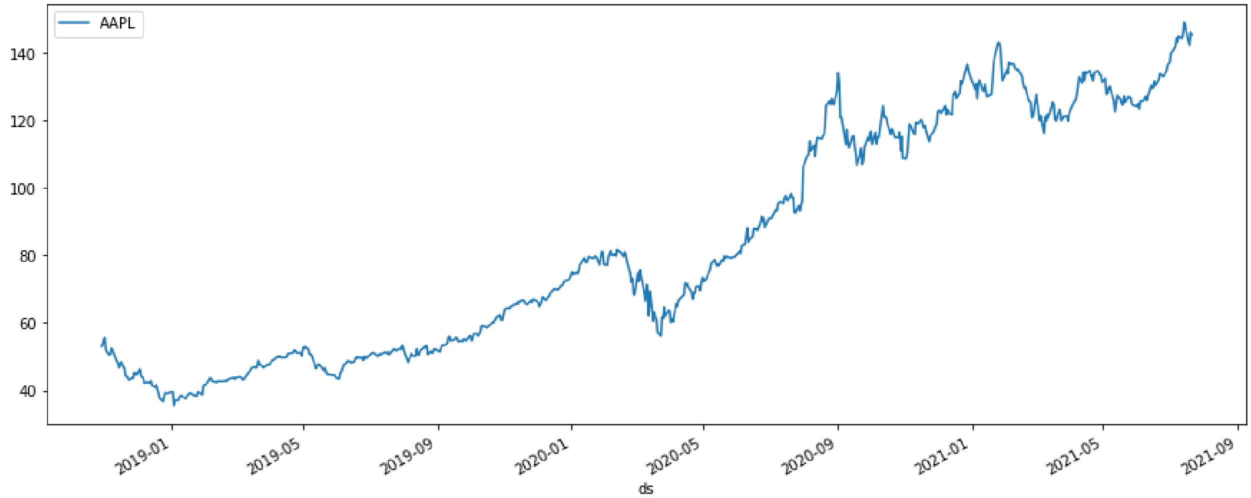
▶ Solution

Visualize the data that we have. We don't want to start making predictions before looking at the past :)

```
# your code here
df.plot(x='ds', y='y', figsize=(15,6), label ='AAPL')
```

```
<Axes: xlabel='ds'>
```

Download



▶ Solution

Finally let's create a new `model` and `fit` (train) it on our DataFrame 🤖

```
# your code here
model = Prophet()
model.fit(df)
```

```
<prophet.forecaster.Prophet at 0x7f223190f7c0>

00:06:54 - cmdstanpy - INFO - Chain [1] start processing
00:06:55 - cmdstanpy - INFO - Chain [1] done processing
```

▶ Solution

In-sample prediction

Let's start with looking at existing data and see how well the model learned the patterns.

Make a `sample` with the last `90 days` of stock prices from our DataFrame `df`

```
# your code here
sample = df[-90:]
sample
```

	y	ds
596	123.99	2021-03-15
597	125.57	2021-03-16
598	124.76	2021-03-17
599	120.53	2021-03-18
600	119.99	2021-03-19
...	...	...
681	148.48	2021-07-15
682	146.39	2021-07-16
683	142.45	2021-07-19
684	146.15	2021-07-20
685	145.40	2021-07-21

90 rows × 2 columns

▶ Solution

Create a `forecast` by using the `.predict()` method of our `model`

```
# your code here
forecast = model.predict(sample)
forecast
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	ad
0	2021-03-15	130.233734	118.899592	127.422310	130.233734	130.233734	-7.198091	-7.198091	-7
1	2021-03-16	130.329189	118.816482	126.934196	130.329189	130.329189	-7.377212	-7.377212	-7
2	2021-03-17	130.424644	118.448904	126.856114	130.424644	130.424644	-7.538293	-7.538293	-7
3	2021-03-18	130.520100	118.628171	126.559920	130.520100	130.520100	-7.931815	-7.931815	-7
4	2021-03-19	130.615555	118.170717	126.437750	130.615555	130.615555	-8.340656	-8.340656	-8
...	...	...	...	...	...	...	...	...	...
85	2021-07-15	141.879272	137.307371	145.434458	141.879272	141.879272	-0.547747	-0.547747	-0
86	2021-07-16	141.974727	137.004513	145.123892	141.974727	141.974727	-0.741454	-0.741454	-0
87	2021-07-19	142.261093	137.607554	145.699860	142.261093	142.261093	-0.782276	-0.782276	-0
88	2021-07-20	142.356548	137.327538	145.910529	142.356548	142.356548	-0.727223	-0.727223	-0
89	2021-07-21	142.452003	137.780307	145.690977	142.452003	142.452003	-0.674497	-0.674497	-0

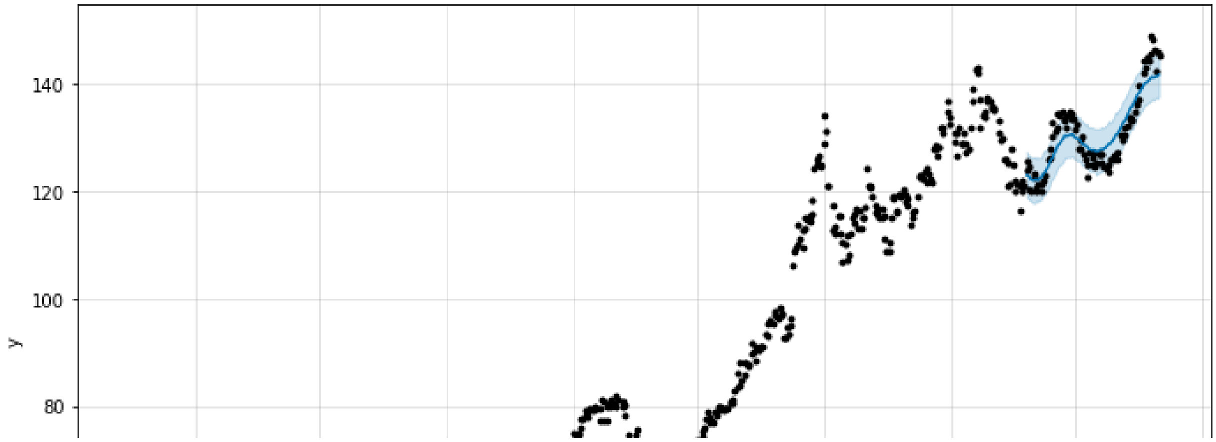
90 rows × 19 columns

▶ Solution

We can look inside the `forecast` variable, but it's not easy to read. Let's **visualize** our forecast instead.

```
# your code here
model.plot(forecast);
```

Download



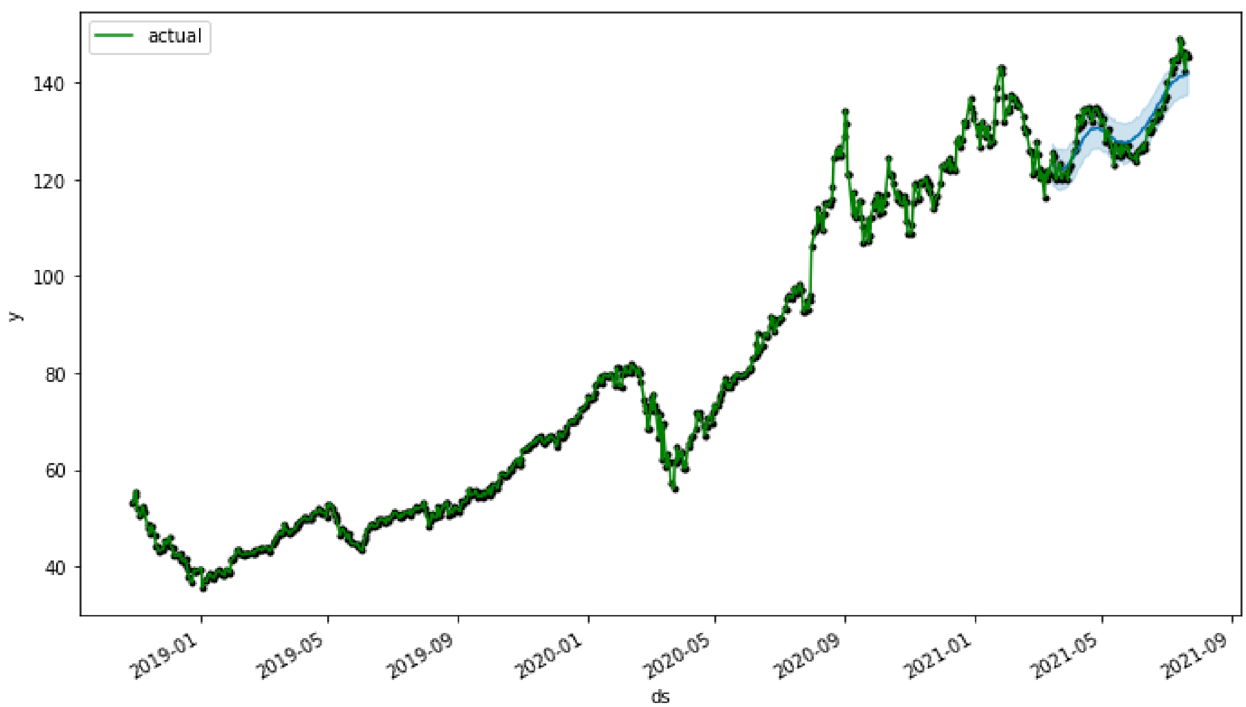
▶ Solution

[Bonus 🏋️] Let's also **plot a line** of the real historic data between the dots. Don't hesitate to check how we did that with the car sales challenge.

```
# your code here
ax = model.plot(forecast).gca()
df.plot(ax=ax,x='ds', y='y', label = 'actual', color = 'g')
```

```
<Axes: xlabel='ds', ylabel='y'>
```

Download



▶ Solution

Finally, let's count the difference between the `y` column in our `sample` and the `yhat` column in our `forecast`

```
# your code here
difference = forecast['yhat'].values - sample['y'].values
np.absolute(difference).mean()
```

```
2.7489457163612547
```

▶ Solution

We can see our predictions are very close - on average just about 25 cents off! Now let's move on to **future predictions** 🔮

Out-of-sample prediction

First, let's create a `future` DataFrame which will contain dates for the next 180 days.

In the previous challenge we had to set our `freq`uency to `MS`, because our car sales were monthly. The only difference here, is that we need to change the `freq` option to `D`, for 'days'.

```
# your code here
future = model.make_future_dataframe(freq='D', periods=90)
future
```

	ds
0	2018-10-29
1	2018-10-30
2	2018-10-31
3	2018-11-01
4	2018-11-02
...	...
771	2021-10-15
772	2021-10-16
773	2021-10-17
774	2021-10-18
775	2021-10-19

776 rows × 1 columns

▶ Solution

[But wait ❗] The stock exchange is **closed on the weekends**, so if we want to be more accurate, we should also remove weekends from our `future` dates. For that we need to do some filtering using the `pandas.datetime.dayofweek` function. Simply run the cell below, to update your `future` DataFrame 😉

```
future = future[future['ds'].dt.dayofweek < 5]
future.tail(10) # last 10 rows, note the weekend gaps
```

	ds
762	2021-10-06
763	2021-10-07
764	2021-10-08
767	2021-10-11
768	2021-10-12
769	2021-10-13
770	2021-10-14
771	2021-10-15
774	2021-10-18
775	2021-10-19

Time to make a `future_forecast` using the `.predict()` method of our `model`

```
# Your code here
future_forecast= model.predict(future)
future_forecast.tail()
```

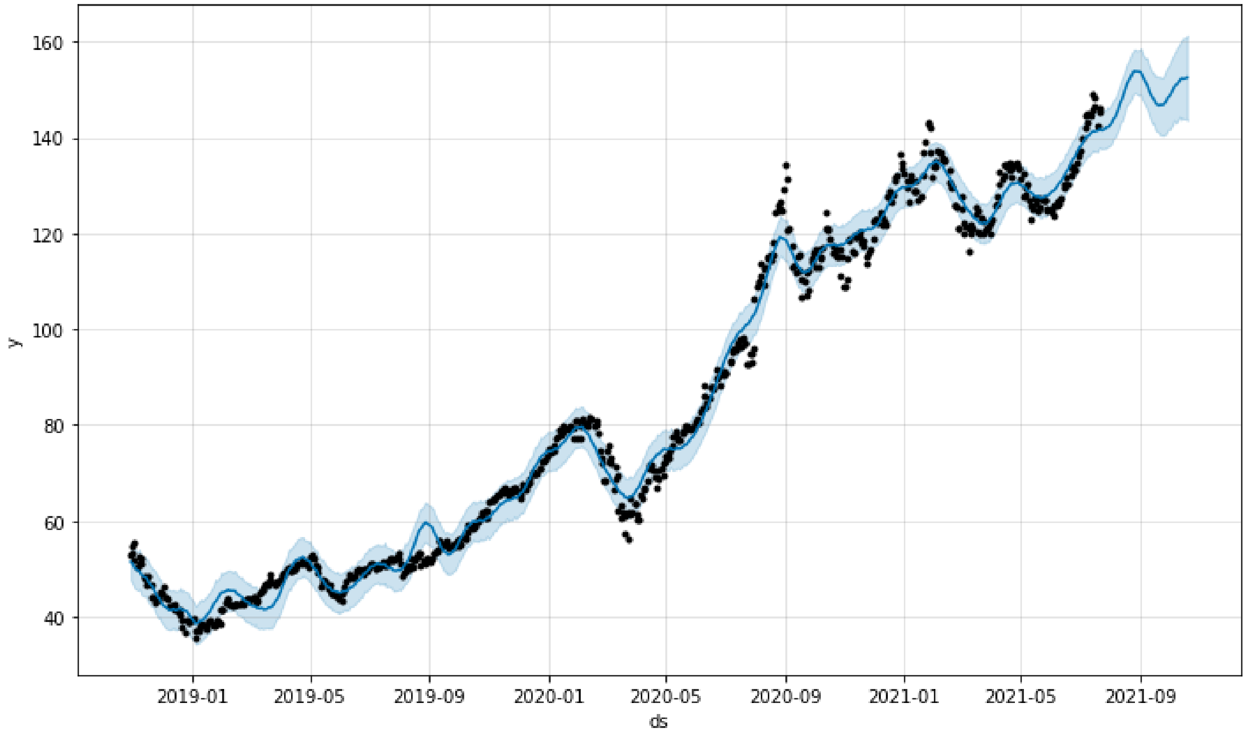
	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	a
745	2021-10-13	150.470243	144.242729	160.362817	143.559098	157.214573	1.921781	1.921781	1
746	2021-10-14	150.565698	143.942407	160.739249	143.482552	157.497882	1.792610	1.792610	1
747	2021-10-15	150.661153	143.908052	160.792199	143.344350	157.661192	1.598304	1.598304	1
748	2021-10-18	150.947519	143.989820	160.899913	143.438201	158.141553	1.491048	1.491048	1
749	2021-10-19	151.042974	143.619372	161.391017	143.354299	158.592615	1.505418	1.505418	1

▶ Solution

Again, looking at this huge DataFrame is not ideal - let's **visualize our predictions**

```
# Your code here
model.plot(future_forecast);
```

Download



▶ Solution

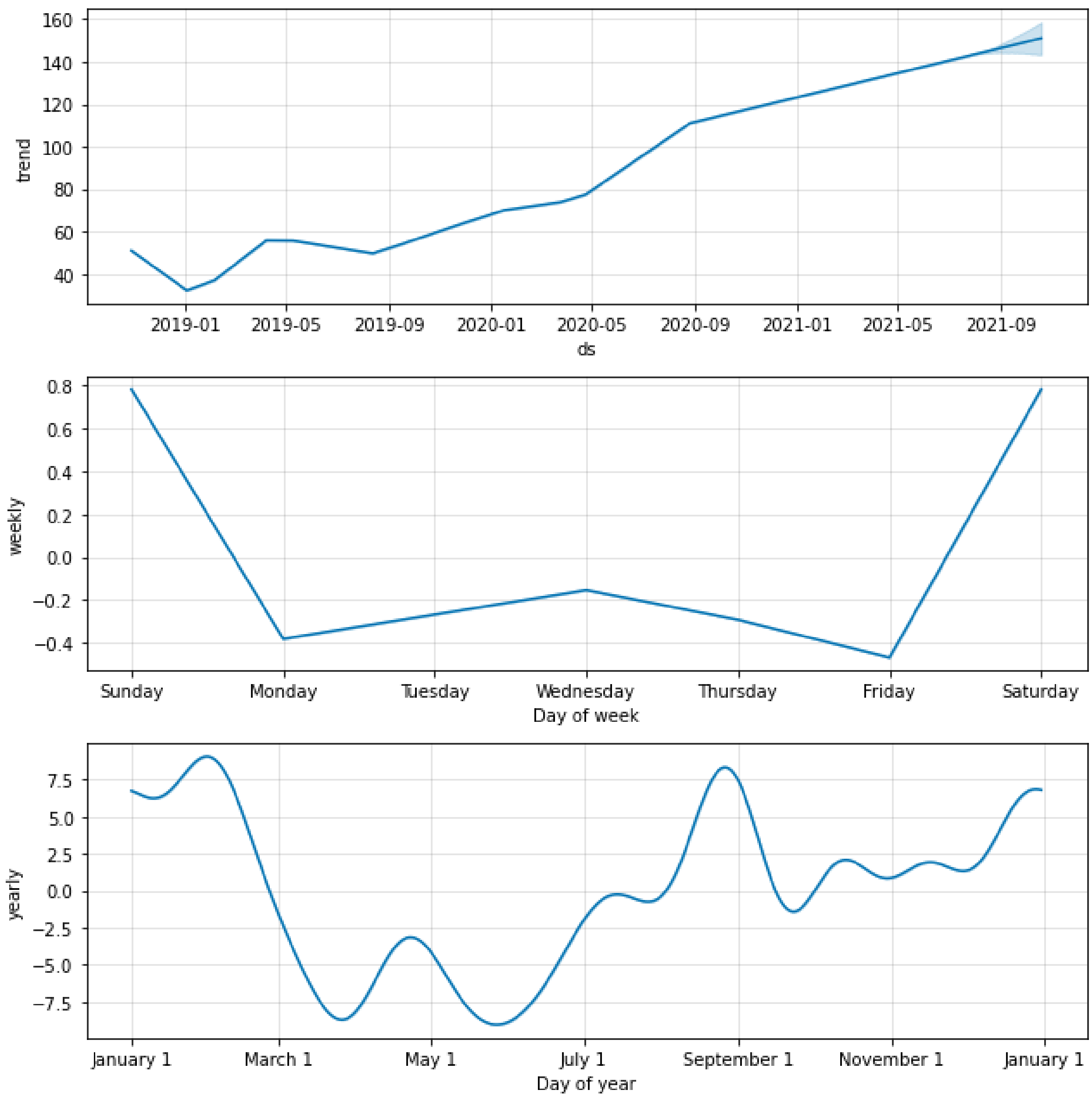
Interesting results! We can clearly see there's a point where our model starts "losing confidence". Let's explore our findings further.

Exploring our Prediction

Let's start by looking at the different **components** of our time series prediction - such as **seasonality** and **trend**. Don't hesitate to check the car sales notebook for the answers!

```
# Your code here
model.plot_components(future_forecast);
```

Download



▶ Solution

Any day traders here? 🙋‍♂️ 🙋‍♀️ Because our data is daily, you can also see the **weekday trends**.

[Bonus 🏋️] Shall we make our graph interactive? Remember the `.plot_plotly()` function we used in our car sales livecode.

```
# Your code here
from prophet.plot import plot_plotly

plot_plotly(model,future_forecast)
```

▶ Solution

Evaluating our Model

Let's use the Diagnostics library from `prophet` to validate our model using `cross_validation`. Run the cell below to import the library first:

```
from prophet.diagnostics import cross_validation
```

Now **create a** `df_cv` DataFrame that is the result of running `cross_validation` on our model, with a horizon of 180, 90 or 60 days - your choice!

The less days you choose, the longer it will take, and the more learning the model will do, because it will chop up your data into those blocks.

```
# Your code here
df_cv = cross_validation(model,horizon ='60 days')
```

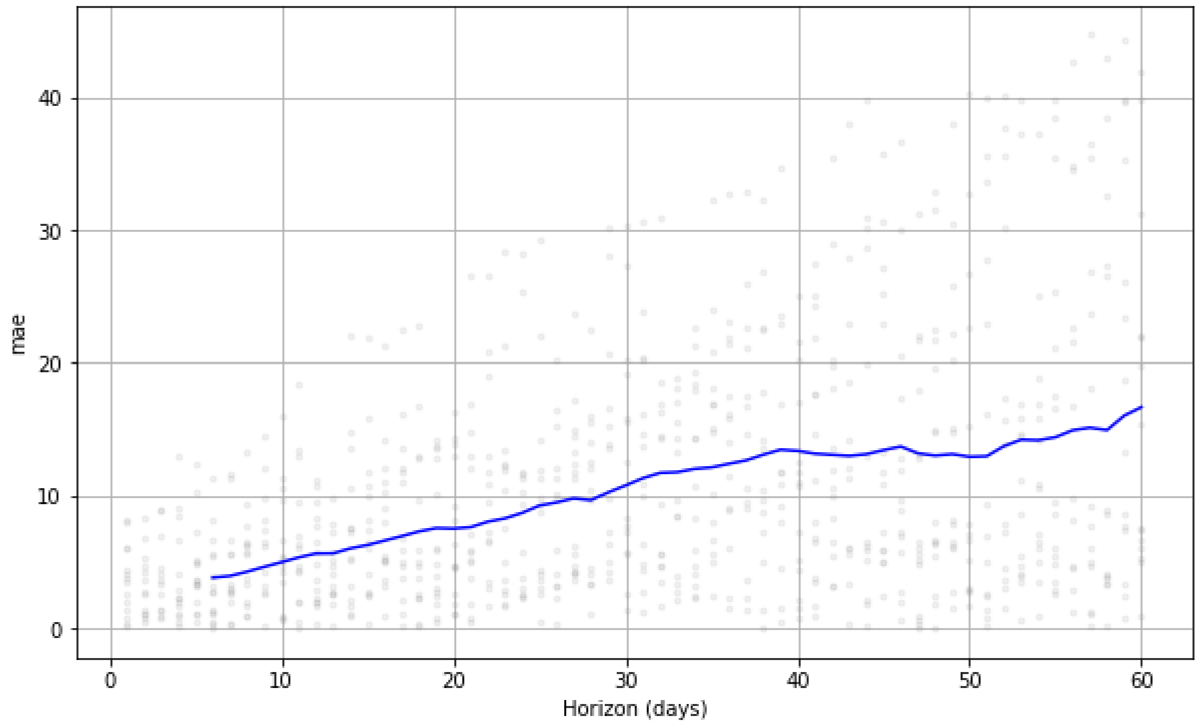
```
00:25:58 - cmdstanpy - INFO - Chain [1] start processing
00:25:58 - cmdstanpy - INFO - Chain [1] done processing
00:25:58 - cmdstanpy - INFO - Chain [1] start processing
00:25:59 - cmdstanpy - INFO - Chain [1] done processing
00:25:59 - cmdstanpy - ERROR - Chain [1] error: error during processing Operation not p
Optimization terminated abnormally. Falling back to Newton.
00:25:59 - cmdstanpy - INFO - Chain [1] start processing
00:26:01 - cmdstanpy - INFO - Chain [1] done processing
00:26:01 - cmdstanpy - INFO - Chain [1] start processing
00:26:01 - cmdstanpy - INFO - Chain [1] done processing
00:26:01 - cmdstanpy - INFO - Chain [1] start processing
00:26:01 - cmdstanpy - INFO - Chain [1] done processing
00:26:01 - cmdstanpy - INFO - Chain [1] start processing
00:26:02 - cmdstanpy - INFO - Chain [1] done processing
00:26:02 - cmdstanpy - INFO - Chain [1] start processing
00:26:02 - cmdstanpy - INFO - Chain [1] done processing
00:26:02 - cmdstanpy - INFO - Chain [1] start processing
00:26:02 - cmdstanpy - INFO - Chain [1] done processing
00:26:02 - cmdstanpy - INFO - Chain [1] start processing
00:26:02 - cmdstanpy - INFO - Chain [1] done processing
```

▶ Solution

Finally, let's visualize the errors (differences) that between our model prediction and the seen reality. We will use the `mae` (Mean Absolute Error) as the metric, same as with our car sales predictions.

```
from prophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='mae')
```

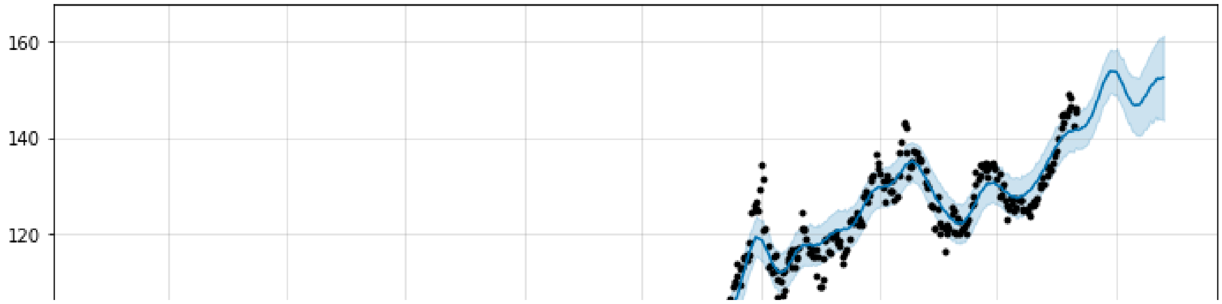
Download



Interpretation

```
model.plot(future_forecast);
```

Download



With the information you have, you can already start making decisions. The rest is up to how risk averse are you and what's your goal!

- We can see from the prediction `plot` that we have a point after which the model quickly starts to lose confidence.
- We can also see the same from the errors - as we try to predict further into the future, our accuracy goes down.
- **But** we can see that typically about up to 30-50 days into the future we are getting good results for 1 hour of work! 💪

Congrats! You now have Python tools for your own predictions! 🔥