032-linear-regression-with-time-series-data

May 5, 2022

Linear Regression with Time Series Data

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
import pytz
from IPython.display import VimeoVideo
from pymongo import MongoClient
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
```

[2]:
```
VimeoVideo("665412117", h="c39a50bd58", width=600)
```

[2]:
```
<IPython.lib.display.VimeoVideo at 0x7f74604338e0>
```

1 Prepare Data

1.1 Import

[3]:
```
VimeoVideo("665412469", h="135f32c7da", width=600)
```

[3]:
```
<IPython.lib.display.VimeoVideo at 0x7f746041e850>
```

Task 3.2.1: Complete to the create a client to connect to the MongoDB server, assign the `"air-quality"` database to `db`, and assign the `"nairobi"` connection to `nairobi`.

- Create a client object for a MongoDB instance.
- Access a database using PyMongo.
- Access a collection in a database using PyMongo.

[4]:
```
client = MongoClient(host = 'localhost', port = 27017)
db = client['air-quality']
nairobi = db['nairobi']
```

[5]:
```
VimeoVideo("665412480", h="c20ed3e570", width=600)
```

[5]:
```
<IPython.lib.display.VimeoVideo at 0x7f73869d4970>
```

Task 3.2.2: Complete the `wrangle` function below so that the `results` from the database query are read into the DataFrame `df`. Be sure that the index of `df` is the `"timestamp"` from the results.

- Create a DataFrame from a dictionary using pandas.

[49]:
```
def wrangle(collection):
    results = collection.find(
        {"metadata.site": 29, "metadata.measurement": "P2"},
        projection={"P2": 1, "timestamp": 1, "_id": 0},
    )

    df = pd.DataFrame(results).set_index('timestamp')

    df.index = df.index.tz_localize('UTC').tz_convert('Africa/Nairobi')

    df = df[df['P2'] < 500]

    df = df['P2'].resample('1H').mean().fillna(method = 'ffill').to_frame()

    df['P2.L1'] = df['P2'].shift(1)

    df.dropna(inplace = True)

    return df
```

[7]:
```
VimeoVideo("665412496", h="d757475f7c", width=600)
```

[7]:
```
<IPython.lib.display.VimeoVideo at 0x7f73869d4df0>
```

Task 3.2.3: Use your wrangle function to read the data from the `nairobi` collection into the DataFrame `df`.

[50]:
```
df = wrangle(nairobi)
print(df.shape)
df.head(10)
```

```
(2927, 2)
```

[50]:

timestamp	P2	P2.L1
2018-09-01 04:00:00+03:00	15.800000	17.541667
2018-09-01 05:00:00+03:00	11.420000	15.800000
2018-09-01 06:00:00+03:00	11.614167	11.420000
2018-09-01 07:00:00+03:00	17.665000	11.614167
2018-09-01 08:00:00+03:00	21.016667	17.665000
2018-09-01 09:00:00+03:00	22.589167	21.016667
2018-09-01 10:00:00+03:00	18.605833	22.589167
2018-09-01 11:00:00+03:00	14.022500	18.605833
2018-09-01 12:00:00+03:00	13.150000	14.022500

```
2018-09-01 13:00:00+03:00  12.806667  13.150000
```

[9]:
```
# Check your work
assert any([isinstance(df, pd.DataFrame), isinstance(df, pd.Series)])
assert len(df) <= 32907
assert isinstance(df.index, pd.DatetimeIndex)
```

[10]:
```
VimeoVideo("665412520", h="e03eefff07", width=600)
```

[10]:
```
<IPython.lib.display.VimeoVideo at 0x7f73869d46a0>
```

Task 3.2.4: Add to your `wrangle` function so that the `DatetimeIndex` for `df` is localized to the correct timezone, `"Africa/Nairobi"`. Don't forget to re-run all the cells above after you change the function.

- Localize a timestamp to another timezone using pandas.

[11]:
```
# Check your work
assert df.index.tzinfo == pytz.timezone("Africa/Nairobi")
```

1.2 Explore

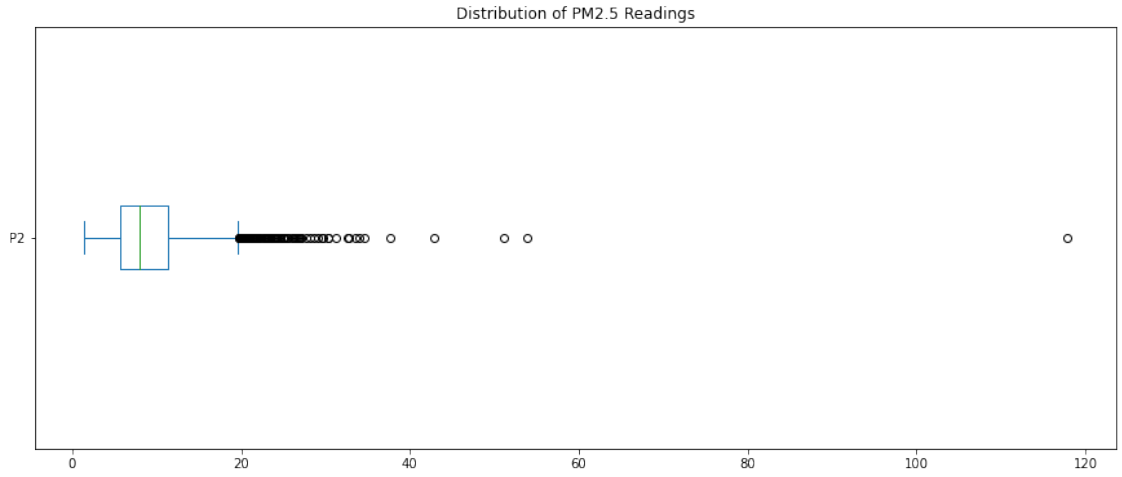
[12]:
```
VimeoVideo("665412546", h="97792cb982", width=600)
```

[12]:
```
<IPython.lib.display.VimeoVideo at 0x7f73869f1cd0>
```

Task 3.2.5: Create a boxplot of the `"P2"` readings in `df`.

- Create a boxplot using pandas.

[29]:
```
fig, ax = plt.subplots(figsize=(15, 6))
df['P2'].plot(kind = 'box', vert = False, title = 'Distribution of PM2.5␣
↪Readings', ax  =ax);
```



[14]:
```
VimeoVideo("665412573", h="b46049021b", width=600)
```

[14]:
```
<IPython.lib.display.VimeoVideo at 0x7f73850d6130>
```

Task 3.2.6: Add to your `wrangle` function so that all `"P2"` readings above 500 are dropped from the dataset. Don't forget to re-run all the cells above after you change the function.

- Subset a DataFrame with a mask using pandas.

[15]:
```
# Check your work
assert len(df) <= 32906
```

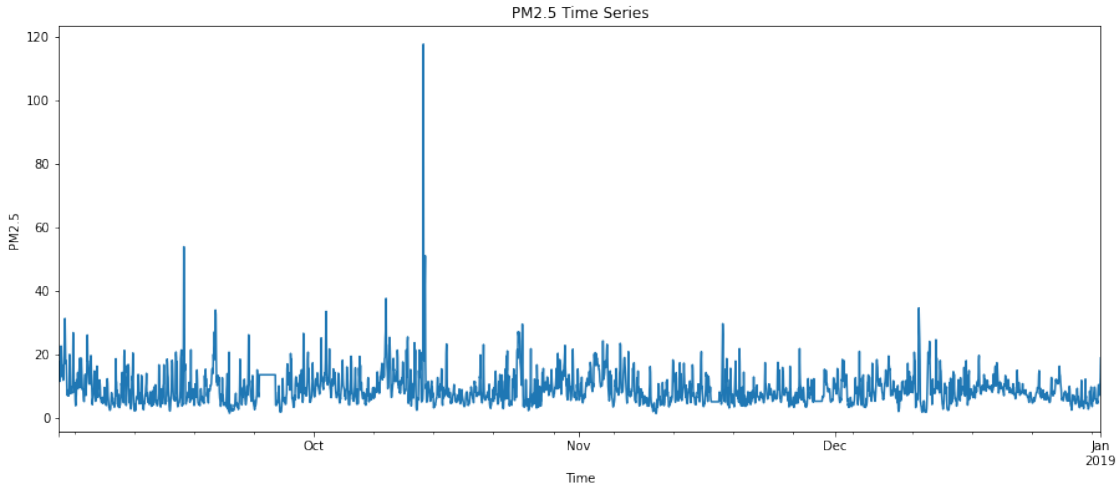
[17]:
```
VimeoVideo("665412594", h="e56c2f6839", width=600)
```

[17]:
```
<IPython.lib.display.VimeoVideo at 0x7f73869d4a90>
```

Task 3.2.7: Create a time series plot of the `"P2"` readings in `df`.

- Create a line plot using pandas.

[30]:
```
fig, ax = plt.subplots(figsize=(15, 6))
df['P2'].plot(xlabel = 'Time', ylabel = 'PM2.5', title = 'PM2.5 Time Series',␣
↪ax = ax);
```



[20]:
```
VimeoVideo("665412601", h="a16c5a73fc", width=600)
```

[20]:
```
<IPython.lib.display.VimeoVideo at 0x7f7384e95a00>
```

Task 3.2.8: Add to your `wrangle` function to resample `df` to provide the mean `"P2"` reading for each hour. Use a forward fill to impute any missing values. Don't forget to re-run all the cells

above after you change the function.

- Resample time series data in pandas.
- Impute missing time series values using pandas.

[26]:
```
df['P2'].resample('1H').mean().fillna(method = 'ffill').to_frame().head()
```

[26]:
```
                                P2
timestamp
2018-09-01 03:00:00+03:00  17.541667
2018-09-01 04:00:00+03:00  15.800000
2018-09-01 05:00:00+03:00  11.420000
2018-09-01 06:00:00+03:00  11.614167
2018-09-01 07:00:00+03:00  17.665000
```

[31]:
```
# Check your work
assert len(df) <= 2928
```

[32]:
```
VimeoVideo("665412649", h="d2e99d2e75", width=600)
```

[32]:
```
<IPython.lib.display.VimeoVideo at 0x7f7375d98550>
```

Task 3.2.9: Plot the rolling average of the `"P2"` readings in `df`. Use a window size of `168` (the number of hours in a week).

- What's a rolling average?
- Calculate a rolling average in pandas.
- Create a line plot using pandas.

[35]:
```
df['P2'].rolling(168).mean() # rolling average - period 'week'
```

[35]:
```
timestamp
2018-09-01 03:00:00+03:00         NaN
2018-09-01 04:00:00+03:00         NaN
2018-09-01 05:00:00+03:00         NaN
2018-09-01 06:00:00+03:00         NaN
2018-09-01 07:00:00+03:00         NaN
                                  ...
2018-12-31 22:00:00+03:00    6.932995
2018-12-31 23:00:00+03:00    6.935927
2019-01-01 00:00:00+03:00    6.938348
2019-01-01 01:00:00+03:00    6.973928
2019-01-01 02:00:00+03:00    7.043333
Freq: H, Name: P2, Length: 2928, dtype: float64
```

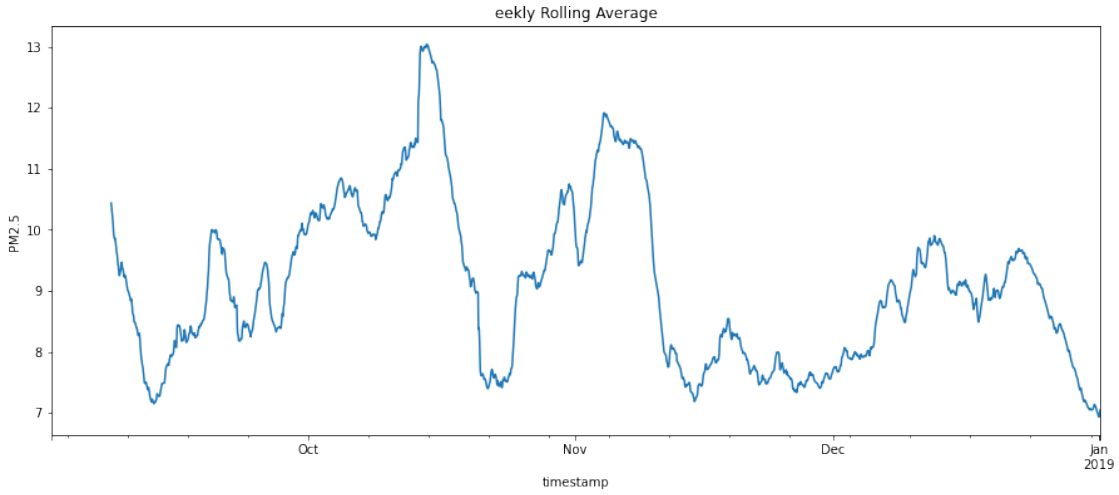
[37]:
```
df['P2'].rolling(168).mean().isnull().sum()
```

[37]:
```
167
```

[36]:
```
len(df)
```

[36]:
```
2928
```

[38]:
```
fig, ax = plt.subplots(figsize=(15, 6))
df["P2"].rolling(168).mean().plot(ax = ax, ylabel = 'PM2.5', title = 'eekly␣
↪Rolling Average');
```



[39]:
```
VimeoVideo("665412693", h="c3bca16aff", width=600)
```

[39]:
```
<IPython.lib.display.VimeoVideo at 0x7f7384a8b850>
```

Task 3.2.10: Add to your `wrangle` function to create a column called `"P2.L1"` that contains the mean`"P2"` reading from the previous hour. Since this new feature will create `NaN` values in your DataFrame, be sure to also drop null rows from `df`.

- Shift the index of a Series in pandas.
- Drop rows with missing values from a DataFrame using pandas.

[51]:
```
# Check your work
assert len(df) <= 11686
assert df.shape[1] == 2
```

[52]:
```
VimeoVideo("665412732", h="059e4088c5", width=600)
```

[52]:
```
<IPython.lib.display.VimeoVideo at 0x7f7372746ac0>
```

Task 3.2.11: Create a correlation matrix for `df`.

- Create a correlation matrix in pandas.

[55]:
```
df.corr()
```

[55]:

	P2	P2.L1
P2	1.000000	0.650679
P2.L1	0.650679	1.000000

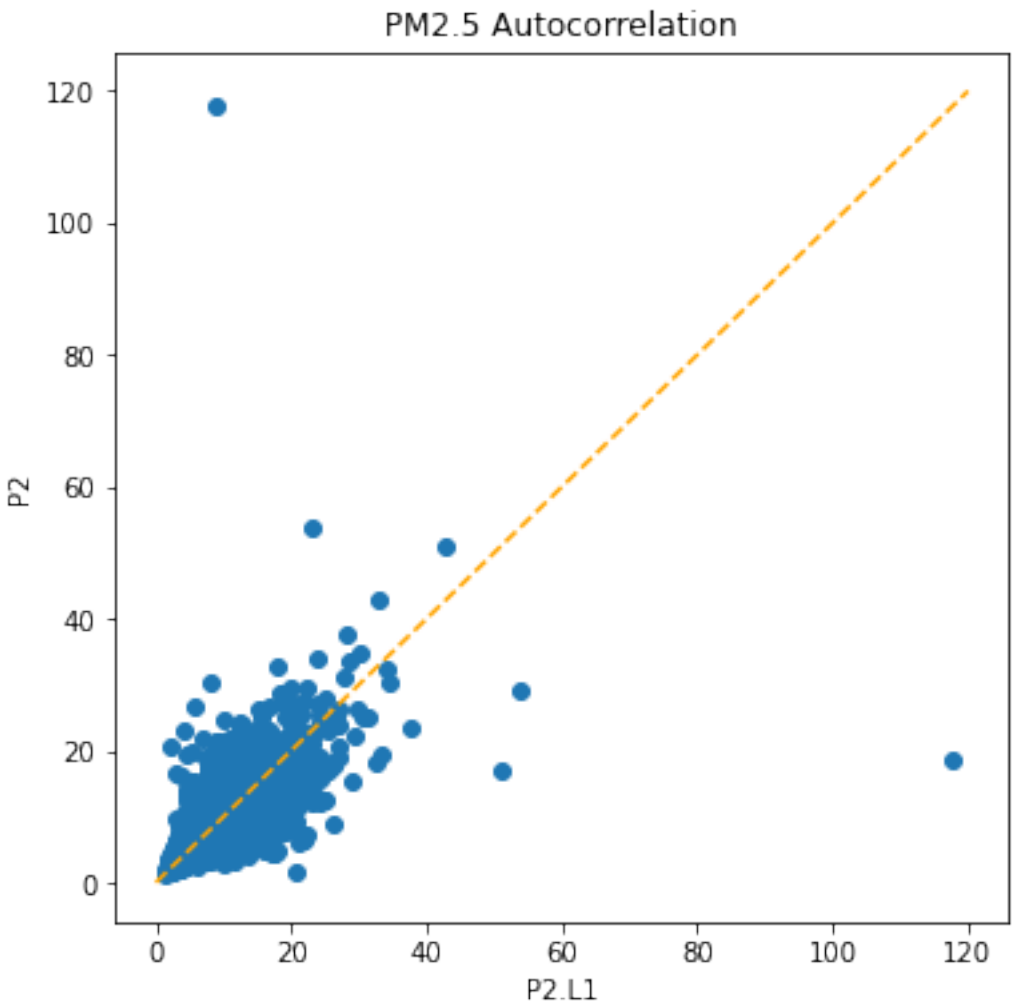
[56]:
```
VimeoVideo("665412741", h="7439cb107c", width=600)
```

[56]:
```
<IPython.lib.display.VimeoVideo at 0x7f73727469a0>
```

Task 3.2.12: Create a scatter plot that shows PM 2.5 mean reading for each our as a function of the mean reading from the previous hour. In other words, `"P2.L1"` should be on the x-axis, and `"P2"` should be on the y-axis. Don't forget to label your axes!

- Create a scatter plot using Matplotlib.

[59]:
```
fig, ax = plt.subplots(figsize=(6, 6))
ax.scatter(x = df['P2.L1'], y = df['P2'])
ax.plot([0, 120], [0, 120], linestyle = '--', color = 'orange')
plt.xlabel('P2.L1')
plt.ylabel('P2')
plt.title('PM2.5 Autocorrelation');
```



1.3 Split

[60]:
```
VimeoVideo("665412762", h="a5eba496f7", width=600)
```

[60]:
```
<IPython.lib.display.VimeoVideo at 0x7f73727125e0>
```

Task 3.2.13: Split the DataFrame `df` into the feature matrix `X` and the target vector `y`. Your target is `"P2"`.

- Subset a DataFrame by selecting one or more columns in pandas.
- Select a Series from a DataFrame in pandas.

[61]:
```
target = "P2"
y = df[target]
X = df.drop(columns = target)
```

[62]:
```
VimeoVideo("665412785", h="03118eda71", width=600)
```

[62]:
```
<IPython.lib.display.VimeoVideo at 0x7f737270b5b0>
```

Task 3.2.14: Split `X` and `y` into training and test sets. The first 80% of the data should be in your training set. The remaining 20% should be in the test set.

- Divide data into training and test sets in pandas.

[64]:
```
cutoff = int(len(X) * 0.8)

X_train, y_train = X.iloc[:cutoff], y.iloc[:cutoff]
X_test, y_test = X.iloc[cutoff:], y.iloc[cutoff:]
```

2 Build Model

2.1 Baseline

Task 3.2.15: Calculate the baseline mean absolute error for your model.

- Calculate summary statistics for a DataFrame or Series in pandas.

[66]:
```
y_pred_baseline = [y.mean()] * len(y)
mae_baseline = mean_absolute_error(y, y_pred_baseline)

print("Mean P2 Reading:", round(y_train.mean(), 2))
print("Baseline MAE:", round(mae_baseline, 2))
```

```
Mean P2 Reading: 9.27
Baseline MAE: 3.65
```

2.2 Iterate

Task 3.2.16: Instantiate a `LinearRegression` model named `model`, and fit it to your training data.

- Instantiate a predictor in scikit-learn.
- Fit a model to training data in scikit-learn.

[67]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

[67]:
```
LinearRegression()
```

2.3 Evaluate

[68]:
```
VimeoVideo("665412844", h="129865775d", width=600)
```

[68]:
```
<IPython.lib.display.VimeoVideo at 0x7f73727197f0>
```

Task 3.2.17: Calculate the training and test mean absolute error for your model.

- Generate predictions using a trained model in scikit-learn.
- Calculate the mean absolute error for a list of predictions in scikit-learn.

[69]:
```
training_mae = mean_absolute_error(y_train, model.predict(X_train))
test_mae = mean_absolute_error(y_test, model.predict(X_test))
print("Training MAE:", round(training_mae, 2))
print("Test MAE:", round(test_mae, 2))
```

```
Training MAE: 2.46
Test MAE: 1.8
```

3 Communicate Results

Task 3.2.18: Extract the intercept and coefficient from your `model`.

- Access an object in a pipeline in scikit-learn

[71]:
```
intercept = model.intercept_
coefficient = model.coef_

print(f"P2 = {intercept} + ({coefficient} * P2.L1)")
```

```
P2 = 3.3555730583913483 + ([0.63789422] * P2.L1)
```

[72]:
```
VimeoVideo("665412870", h="318d69683e", width=600)
```

[72]:
```
<IPython.lib.display.VimeoVideo at 0x7f737271f790>
```

Task 3.2.19: Create a DataFrame `df_pred_test` that has two columns: `"y_test"` and `"y_pred"`. The first should contain the true values for your test set, and the second should contain your model's predictions. Be sure the index of `df_pred_test` matches the index of `y_test`.

- Create a DataFrame from a dictionary using pandas.

[73]:
```
df_pred_test = pd.DataFrame(
    {'y_test': y_test,
     'y_pred': model.predict(X_test)
    }
)
df_pred_test.head()
```

[73]:

timestamp	y_test	y_pred
2018-12-07 17:00:00+03:00	7.070000	8.478927
2018-12-07 18:00:00+03:00	8.968333	7.865485
2018-12-07 19:00:00+03:00	11.630833	9.076421
2018-12-07 20:00:00+03:00	11.525833	10.774814
2018-12-07 21:00:00+03:00	9.533333	10.707836

[74]:
```
VimeoVideo("665412891", h="39d7356a26", width=600)
```

[74]:
```
<IPython.lib.display.VimeoVideo at 0x7f73726e2760>
```

Task 3.2.20: Create a time series line plot for the values in `test_predictions` using plotly express. Be sure that the y-axis is properly labeled as `"P2"`.

- Create a line plot using plotly express.

[75]:
```
fig = px.line(df_pred_test, labels = {'value': 'P2'})
fig.show()
```