1-Wragling Data With MongoDB

Bu çalışma, MongoDB veritabanına bağlanmayı, verileri sorgulamayı ve işlemeyi içeriyor.

1. İlk olarak kütüphaneleri içe aktarıyoruz.

- PrettyPrinter : Bu, veriyi daha okunaklı bir şekilde yazdırmak için kullanılır.
- Pandas: Veri analizi ve işleme için kullanılır.
- MongoClient: MongoDB veritabanına bağlanmak için kullanılır.

```
from pprint import PrettyPrinter

import pandas as pd

from pymongo import MongoClient
```

2. PrettyPrinter ile bir nesne oluşturuyoruz ve bu nesne, verileri düzenli bir şekilde yazdırmak için kullanılacak. Girintileri 2 birim olarak ayarlıyoruz.

```
pp = PrettyPrinter(indent=2)
```

3. MongoClient kullanarak bir MongoDB veritabanına bağlanıyoruz. `host` ve `port` parametreleri ile MongoDB sunucuya bağlanıyoruz. Bu adım, veritabanına erişmek için gerekli olan istemciyi oluşturur.

```
client = MongoClient(host="localhost",port=27017)
```

4. MongoDB sunucudaki mevcut veritabanlarını listeliyoruz ve bu bilgiyi düzenli bir şekilde yazdırıyoruz.

```
pp.pprint(list(client.list_databases()))
```

5. "air-quality" adlı bir veritabanına erişmek için bu veritabanına bir referans oluşturuyoruz.

```
db = client["air-quality"]
```

6. Veritabanındaki koleksiyonları listeliyoruz ve döngü kullanarak her koleksiyonun adını yazdırıyoruz.

```
for c in db.list_collections():
    print(c["name"])
```

7. "nairobi" koleksiyonuna erişmek için bir referans oluşturuyoruz.

```
nairobi = db["nairobi"]
```

8. "nairobi" koleksiyonundaki belge sayısını sayıyoruz ve bu koleksiyondaki belgelerin toplam sayısını alıyoruz.

```
nairobi.count_documents({})
```

9. "nairobi" koleksiyonundan bir belgeyi alıyoruz ve bu belgeyi yazdırıyoruz. Bu, veritabanındaki bir belgeye erişim sağlar.

```
result = nairobi.find_one({})
pp.pprint(result)
```

10. "nairobi" koleksiyonundaki "metadata.site" alanındaki benzersiz değerleri alıyorsunuz ve bu değerleri yazdırıyorsunuz.

```
nairobi.distinct("metadata.site")
```

11. Bu id ye sahip olan belgelerin sayısını hesaplıyoruz ve bu sayıları yazdırıyoruz.

```
print("Documents from site 6:", nairobi.count_documents({"metadata.site" : 6}))
print("Documents from site 29:", nairobi.count_documents({"metadata.site" : 29}))
```

12. "result" adlı değişkeni kullanarak "metadata.site" alanına göre gruplama yaparak belge sayılarını hesaplıyoruz.

```
result = nairobi.aggregate(
    [
        {"$group": {"_id": "$metadata.site", "count" : {"$count": {}}}}
    ]
)
pp.pprint(list(result))
```

13. `"metadata.measurement"` alanındaki benzersiz değerleri alıyoruz ve bu değerleri yazdırıyoruz. Bu kod buradaki tüm farklı ölçüm türlerini liste olarak döndürüyor.

```
nairobi.distinct("metadata.measurement")
```

14. "P2" ölçümünü içeren belgeleri kullanacağımız için bunları alıyoruz ve ilk 3 ölçümü liste halinde yazdırıyoruz.

```
result = nairobi.find({"metadata.mesarument": "P2"}).limit(3)
pp.pprint(list(result))
```

15. "metadata.site" değer 29 olan belgeler için gruplama yaparak belge sayılarını hesaplıyoruz.

```
result = nairobi.aggregate(
    [
        {"$match": {"metadata.site": 29}},
        {"$group": {"_id": "$metadata.measurement", "count": {"$count": {}}}}
    ]
)
pp.pprint(list(result))
```

16. "metadata.site" değeri 6 olan belgeler için gruplama yaparak belge sayılarını hesaplıyoruz.

```
result = nairobi.aggregate(
    [
        {"$match": {"metadata.site" :6 }},
        {"$group": {"_id": "$metadata.measurement", "count" : {"$count": {}}}}
    ]
)
pp.pprint(list(result))
```

17. "metadata.site" değeri 29 ve "metadata.measurement" değeri "P2" olan belgelerin "P2" ve "timestamp" alanlarını alıyoruz ve bu verileri yazdırıyoruz.

```
result = nairobi.find(
    {"metadata.site": 29, "metadata.measurement":"P2"},
    projection={"P2":1, "timestamp":1, "_id":0}

)
pp.pprint(result.next())
```

18. Son olarak, bu verileri bir DataFrame'e dönüştürüyoruz ve bu DataFrame'in ilk beş satırını yazdırıyoruz.

```
df = pd.DataFrame(result).set_index("timestamp")
df.head()
```

2- Linear Regression With Time Series Data

Bu çalışma, elimizde bulunan ölçümü yapılmış hava kalitesi verilerini işleme ve analiz etme işlemlerini içeriyor.

1. Kütüphaneleri içe aktarıyoruz:

- matplotlib.pyplot: Verileri görselleştirmek için kullanılır.

- pandas: Veri analizi ve işleme için kullanılır.

- plotly.express: İnteraktif grafikler oluşturmak için kullanılır.

- pytz: Zaman dilimi (timezone) işlemleri için kullanılır.

- IPython.display: Jupyter Notebook veya IPython ortamında videoları göstermek için kullanılır.

- pymongo: MongoDB veritabanı ile etkileşim kurmak için kullanılır.

- sklearn.linear_model: Doğrusal regresyon modelini eğitmek için kullanılır.

- sklearn.metrics: Model performansını değerlendirmek için kullanılır.

2. Bir MongoDB veritabanına bağlantı kurarız:

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

3. `wrangle` adlı bir fonksiyon tanımlarız. Bu fonksiyon, veritabanından belirli verileri sorgular ve işler.

- `collection.find()`: MongoDB koleksiyonundan belirli kriterlere sahip belgeleri sorgular.
  - Verileri bir DataFrame'e dönüştürürüz.
  - Zaman damgalarını düzgün bir zaman dilimine dönüştürürüz.
  - PM2.5 ölçüm değeri (P2) 500'den küçük olanları filtreleriz.
  - Saatlik örnekleme ile ortalama alırız ve eksik değerleri doldururuz.
  - Bir önceki saatteki değeri (P2.L1) hesaplarız.

```
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
```

4. `wrangle` fonksiyonunu kullanarak verileri işler ve `df` adlı bir DataFrame oluştururuz. Veri çerçevesinin şeklini ve ilk 10 satırını görüntüleriz.

```
df = wrangle(nairobi)
print(df.shape)
df.head(10)
```

5. PM2.5 okumalarının dağılımını gösteren bir kutu grafiği oluştururuz.

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

6. PM2.5 ölçümlerinin zaman serisini çizen bir çizgi grafiği oluştururuz.

```
fig, ax = plt.subplots(figsize=(15, 6))
df["P2"].plot(xlabel="Time", ylabel="PM2.5", title="PM2.5 Time Series", ax=ax)
```

7. Haftalık kayan ortalama grafiği oluştururuz.

```
df["P2"].rolling(168).mean()
fig, ax = plt.subplots(figsize=(15, 6))
df["P2"].rolling(168).mean().plot(ax=ax, ylabel="PM2.5", title="Weekly Rolling Average")
```

8. Veriler arasındaki korelasyonu hesaplarız.

```
df.corr()
```

9. PM2.5 değerlerinin otokorelasyonunu gösteren bir dağılım grafiği oluştururuz.

```
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")
```

10. Verilerimizin %80’ ini eğitim verisi ve %20’sini test verisi olacak şekilde ayırırız.

```
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:]
```

11. Veri setimizin uzunluğunu doğrularız.

```
len(X_train) + len(X_test) == len(X)
```

12. y_pred_baseline isimli değişkene eğitim verisinin ortalama değerini atıyoruz. mae_baseline isimli değişkene ise eğitim verisi ile eğitim verisinin ortalama değeri arasındaki farkı hesaplayıp atıyoruz.

```
y_pred_baseline = y_train.mean()
mae_baseline = abs(y_train - y_pred_baseline).mean()

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

13. Lineer regresyon modeli oluşturup eğitiyoruz.

```
model = LinearRegression()
model.fit(X_train, y_train)
```

14. Training_mae değişkeni , eğitim verisi üzerinde modelin mae'sini görüntüler. Test_mae değişkeni ise test verisi üzerinde modelin mae'sini görüntüler.

```
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))
```

15. intercept ve coefficient ifadeleri, lineer regresyon modelinin denklemi olan P2=intercept + (coefficient*P2.L1)'i görüntüler.

```
intercept = model.intercept_.round(2)
coefficient = model.coef_.round(2)[0]

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

16. df_pred_test adlı bir veri çerçevesi oluşturulur, bu çerçeve eğitim verileri üzerindeki gerçek ve tahmin edilen PM2.5 değerlerini içerir.

```
df_pred_test = pd.DataFrame(
{
    "y_test":y_test,
    "y_pred":model.predict(X_test)


}

)
df_pred_test.head()
```

17. Bu veri çerçevesi kullanılarak bir çizgi grafiği oluşturulur ve görüntülenir. Bu grafik, gerçek ve tahmin edilen PM2.5 değerlerini görselleştirir.

```
fig = px.line(df_pred_test, labels={"value":"P2"})
fig.show()
```

3- AutoRegressive Model

Bu çalışma, zaman serisi verileri kullanarak otoregresif (AR) bir model oluşturmayı, bu modeli eğitmeyi, test etmeyi ve ardışık tahminler yapmayı göstermektedir.

1. İlk olarak, gerekli kütüphaneleri içe aktarıyoruz. Bu kütüphaneler, veri analizi, zaman serisi analizi ve model oluşturma işlemleri için kullanılacaktır.

```
import warnings

import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
from IPython.display import VimeoVideo
from pymongo import MongoClient
from sklearn.metrics import mean_absolute_error
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.ar_model import AutoReg

warnings.simplefilter(action="ignore", category=FutureWarning)
```

2. Bu çalışmada kullanacağımı veriyi seçiyoruz ve db adlı değişkene atıyoruz.

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

3. "wrangle" adlı bir işlev oluşturuyoruz. Bu işlev, veriyi MongoDB'den alır, temizler ve zaman serisi olarak düzenler. Veriyi çekerken "metadata.site" değeri 29 ve "metadata.measurement" değeri "P2" olan verileri alır. Veriyi bir saatlik pencereye yeniden örnekler, eksik değerleri önceki değerlerle doldurur ve son olarak "y" adlı bir Seri olarak döndürür.

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

    # Read data into DataFrame
    df = pd.DataFrame(list(results)).set_index("timestamp")

    # Localize timezone
    df.index = df.index.tz_localize("UTC").tz_convert("Africa/Nairobi")

    # Remove outliers
    df = df[df["P2"] < 500]

    # Resample to 1hr window
    y = df["P2"].resample("1H").mean().fillna(method='ffill')

    return y
```

4. "y" Serisi, oluşturduğumuz zaman serisi veriyi temsil eder.

```
y = wrangle(nairobi)
y.head()
```

5. Ardından, otokorelasyon fonksiyonu (ACF) grafiği oluştururuz. ACF grafiği, zaman serisinin kendi gecikmeli değerleri ile nasıl ilişkilendiğini gösterir. Bu grafik, gecikmeli korelasyonu görselleştirir.

```
fig, ax = plt.subplots(figsize=(15, 6))
plot_acf(y, ax=ax)
plt.xlabel("Lag [hours]")
plt.ylabel("Correlation Coefficient");
```

6. Burada, bir saat önceki değer ile iki saat önceki değer arasındaki ilişkiyi kontrol ederiz.

```
y.shift(1).corr(y.shift(2))
```

7. Zaman serisindeki gecikmeler arasındaki parsiyel otokorelasyon ilişkisini görselleştiren bir grafik oluştururuz. Bu grafik, zaman serisinin daha önceki saatler veya günler arasındaki etkileşimini incelemek ve zaman serisinin otokorelasyon yapısını anlamak için kullanılır.

```

```

8. Ardından, kırpma sınırını ("cutoff_test") hesaplarız. Veriyi eğitim ve test kümelerine ayırmak için kullanılır. Sonrasında eğitim ve test kümelerini tanımlarız.

```
cutoff_test = int(len(y) * 0.95)

y_train = y.iloc[:cutoff_test]
y_test = y.iloc[cutoff_test:]
```

9. Temel hatanın (baseline mean absolute error) hesaplanması için "y_train_mean" ve "y_pred_baseline" değişkenleri oluşturulur. Temel hata, sadece ortalama bir tahmin kullanılarak hesaplanan ortalama mutlak hata değeridir.

```
y_train_mean = y_train.mean()
y_pred_baseline = [y_train_mean] * len(y_train)
mae_baseline = mean_absolute_error(y_train, y_pred_baseline)

print("Mean P2 Reading:", round(y_train_mean, 2))
print("Baseline MAE:", round(mae_baseline, 2))
```

10. "AutoReg" modelini oluştururuz ve eğitiriz. Bu, otoregresif bir model oluşturur ve eğitim verisi üzerine uyarlar.

```
model = AutoReg(y_train, lags=26).fit()
```

11. Modelin tahminleri ile gerçek veriler arasındaki ortalama mutlak hatayı hesaplarız.

```
y_pred = model.predict().dropna()
training_mae = mean_absolute_error(y_train.iloc[26:], y_pred)
print("Training MAE:", training_mae)
```

12. Eğitim verileri ile modelin tahminleri arasındaki farkları temsil eden hata serisini oluştururuz.

```
y_train_resid = y_train-y_pred
y_train_resid.tail()
```

13. Bir grafik oluşturarak bir modelin eğitim verileri üzerindeki kalıntılarını (residuals) görsel olarak incelemeyi amaçlar. Kalıntılar, modelin gerçek gözlemler ile tahminleri arasındaki farkları temsil eder ve modelin ne kadar iyi veya kötü uyum sağladığını değerlendirmek için kullanılır.

```
fig, ax = plt.subplots(figsize=(15, 6))
y_train_resid.plot(ylabel="Residual Value", ax=ax)
```

14. Aynı sebeple çizilmiş bir histogram grafiğidir.

```
y_train_resid.hist()
plt.xlabel("Residual Value")
plt.ylabel("Frequency")
plt.title("AR(26), Distribution of Residuals")
```

15. Bir otokorelasyon fonksiyonu oluşturur ve bunu grafik üzerinde gösterir. Otokorelasyon fonksiyonu bir zaman serisinin kendi önceki zaman noktaları ile nasıl ilişkilendiğini gösterir. Bu, zaman serisinin kendine benzer desenlerinin olup olmadığını veya otokorelasyon yapısının ne olduğunu incelemek için kullanılır.

```
fig, ax = plt.subplots(figsize=(15, 6))
plot_acf(y_train_resid, ax=ax)
```

16. Burada, modelin test verileri üzerindeki performansını değerlendirmeye yönelik işlemler yaparız. Daha spesifik olarak, test verileri üzerindeki ortalama mutlak hata değerini hesaplar ve yazdırırız.

```
y_pred_test = model.predict(y_test.index.min(), y_test.index.max())
test_mae = mean_absolute_error(y_test, y_pred_test)
print("Test MAE:", test_mae)
```

17. Modelin tahminlerini ve gerçek değerlerini içeren bir pandas DataFrame oluştururuz.

```
df_pred_test = pd.DataFrame(
    {"y_test": y_test, "y_pred": y_pred_test}, index=y_test.index)
```

18. Burada oluşturduğumuz çizgi grafiği, gerçek değerler ile tahmin edilen değerler arasındaki ilişkiyi görsel olarak gösterir.

```
fig = px.line(df_pred_test, labels={"value": "P2"})
fig.show()
```

19. Burada, otoregresyon modelini kullanarak zaman serisindeki verileri ardışık tahmin eder ve bu tahminleri bi pandasSeries nesnesi olan y_pred_wfv serisine atar. Ayrıca her tahmin sonrası modelin geçmişini günceller.

```
%%capture

y_pred_wfv = pd.Series()
history = y_train.copy()
for i in range(len(y_test)):
    model=AutoReg(history, lags=26).fit()
    next_pred=model.forecast()
    y_pred_wfv=y_pred_wfv.append(next_pred)
    history=history.append(y_test[next_pred.index])
    pass
```

20. Burada yürüme doğrulama tahmin yöntemini kullanırız. Bu yöntemde, modelde her adımda bir sonraki değer tahmin edilir ve bu şekilde ardışık tahminler oluşturulur. Bu tahminleri gerçek değerler ile karşılaştırarak ortalama mutlak hata hesaplarız.

```
test_mae = mean_absolute_error(y_test,y_pred_wfv)
print("Test MAE (walk forward validation):", round(test_mae, 2))
```

21. AR modelinin belirli gecikme değerleri için oluşan otoregresyon katsayılarını yazdırırız.

```
print(model.params)
```

22. Test verileri ile yürüme doğrulama yöntemi ile yapılan tahminlerin görsel olarak karşılaştırmasının yapıldığı bir çizgi grafiği oluşturulur.

```
df_pred_test=pd.DataFrame(
    {"y_test":y_test,"y_pred_wfv":y_pred_wfv}
)
fig = px.line(df_pred_test, labels={"value": "PM2.5"})
fig.show()
```