import pandas as pd

import numpy as np

import pymongo

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

plt.style.use('seaborn-darkgrid')

from sklearn.linear\_model import LinearRegression

myclient = pymongo.MongoClient("mongodb://localhost:27017")

mydb = myclient["golddata"]

mycol = mydb["golddata"]

deger = mycol.find()

list\_deger = list(deger)

df = pd.DataFrame(list\_deger)

Açıklama: Bu kod parçacığında, ilk olarak pandas, numpy, pymongo, seaborn ve matplotlib kütüphanelerini içe aktarıyorsunuz. Daha sonra '%matplotlib inline' komutu ile Jupyter Notebook ortamında grafiklerin çizilebileceğini belirtiyorsunuz ve grafiklerin görünümünü 'seaborn-darkgrid' şablonu kullanarak ayarlıyorsunuz. Daha sonra, sklearn kütüphanesinden Linear Regression (Doğrusal Regresyon) sınıfını içe aktarıyorsunuz.

Sonra, MongoDB veritabanına bağlanmak için pymongo kütüphanesini kullanarak bir MongoClient nesnesi oluşturuyorsunuz. Bu nesnenin veritabanı adı "golddata" olan ve veritabanı içinde "golddata" isimli bir collection'ı olan bir veritabanına bağlanıyorsunuz. Daha sonra, collection içindeki tüm verileri almak için 'find()' metodunu kullanarak değişkene 'deger' atıyorsunuz ve bu değişkeni liste şekline dönüştürüyorsunuz. Son olarak, bu listeyi pandas DataFrame nesnesine dönüştürüyorsunuz ve bu nesneyi 'df' değişkenine atıyorsunuz.

df.set\_index("Date", inplace=True)

Bu kod satırında, pandas DataFrame nesnesinin 'Date' (Tarih) sütununu index olarak ayarlıyorsunuz. 'inplace=True' parametresi ile de bu işlemi 'df' DataFrame nesnesinin kendisine uyguluyorsunuz, yani 'df' nesnesinin kendisinde değişiklik yapılır. Bu işlem sonucunda, 'Date' sütunu artık DataFrame'in indexi olacaktır ve DataFrame'in ilk sütunu olmayacaktır. Örneğin, eğer DataFrame'inizin başlıkları şu şekilde olsun: "Date, Open, High, Low, Close, Volume" o zaman bu işlem sonucunda DataFrame'in başlıkları şu şekilde olacak: "Open, High, Low, Close, Volume" ve index olarak "Date" sütunu kullanılacaktır.

#İlgili olmayan sütunları kaldırıyoruz ve dropna()

#işlevini kullanarak NaN değerlerini düşürüyoruz.

#Ardından Gold ETF kapanış fiyatını çiziyoruz.

df = df[['Close']]

# Drop rows with missing values

df = df.dropna()

# Plot the closing price of GLD

df.Close.plot(figsize=(10, 7),color='r')

plt.ylabel("Altın Fiyatı")

plt.title("Altın Fiyat Serisi")

plt.show()

Bu kod bloğunda, öncelikle 'df' DataFrame nesnesinin içinden sadece 'Close' (Kapanış) sütununu seçiyorsunuz. Daha sonra, 'dropna()' metodunu kullanarak DataFrame nesnesindeki NaN değerlerini düşürüyorsunuz. Bu metod, NaN değerleri olan satırları DataFrame nesnesinden çıkartır.

Daha sonra, 'Close' sütununun değerlerini 'df.Close.plot()' metodu ile grafik çizdiriyorsunuz. 'figsize' parametresi ile grafik boyutunu belirliyorsunuz ve 'color' parametresi ile grafik çizgi rengini belirliyorsunuz. 'plt.ylabel()' ve 'plt.title()' fonksiyonları ile grafik eksenlerinin etiketlerini ve grafik başlığını belirliyorsunuz. Son olarak, 'plt.show()' fonksiyonu ile grafiği gösteriyorsunuz.

Bu kod bloğu, 'Close' sütunundaki değerleri 'Altın Fiyat Serisi' adı altında bir grafikte gösterir ve grafiğin y eksenini 'Altın Fiyatı' olarak etiketler.

#Açıklayıcı değişkenleri tanımlayın

#Açıklayıcı değişken, ertesi gün Altın ETF fiyatının değerini belirlemek için

#manipüleedilen bir değişkendir. Basitçe, Gold ETF fiyatını tahmin etmek için

#kullanmak istediğimiz özelliklerdir.

#Bu stratejideki açıklayıcı değişkenler, son 3 gün ve 9 günün hareketli

#ortalamalarıdır.NaN değerlerini dropna() işlevini kullanarak

#düşürüyoruz ve özellik değişkenlerini X'te saklıyoruz.

#Ancak X'e Gold ETF fiyatlarını tahmin etmede faydalı olduğunu düşündüğünüz

#daha fazla değişken ekleyebilirsiniz. Bu değişkenler teknik göstergeler,

#Gold miners ETF (GDX) veya Oil ETF (USO)

#gibi başka bir ETF'nin fiyatı veya ABD ekonomik verileri olabilir.

#Bağımlı değişkeni tanımla

#Benzer şekilde bağımlı değişken, açıklayıcı değişkenlerin değerlerine bağlıdır.

#Basitçe söylemek gerekirse, tahmin etmeye çalıştığımız Altın ETF fiyatıdır.

#Altın ETF fiyatını y cinsinden saklarız.

df['S\_3'] = df['Close'].rolling(window=3).mean()

df['S\_9'] = df['Close'].rolling(window=9).mean()

df['sonraki\_gün\_fiyatı'] = df['Close'].shift(-1)

df = df.dropna()

X = df[['S\_3', 'S\_9']]

# Define dependent variable

y = df['sonraki\_gün\_fiyatı']

Bu kod bloğunda, Altın ETF fiyatını tahmin etmek için açıklayıcı değişkenler oluşturuyorsunuz. Öncelikle, 'Close' sütununun 3 günlük ve 9 günlük hareketli ortalamalarını oluşturuyorsunuz. Bu hareketli ortalamaları 'S\_3' ve 'S\_9' sütunlarına ekliyorsunuz. Daha sonra, 'Close' sütunundaki değerlerin bir sonraki güne ait değerlerini 'sonraki\_gün\_fiyatı' sütununa ekliyorsunuz. Bu sütunun değerlerini oluşturmak için 'shift(-1)' metodunu kullanıyorsunuz. Bu metod, her bir değerin bir sonraki güne ait değerini verir.

Sonra, NaN değerlerini 'dropna()' metodu ile düşürüyorsunuz ve 'X' değişkenine 'S\_3' ve 'S\_9' sütunlarını seçiyorsunuz. Bu sütunlar, açıklayıcı değişkenleriniz olacaktır. Daha sonra, bağımlı değişkeniniz olarak 'sonraki\_gün\_fiyatı' sütununu 'y' değişkenine atıyorsunuz.

Bu kod bloğu ile açıklayıcı değişkenlerinizi ve bağımlı değişkeninizi oluşturmuş olursunuz. Bu değişkenleri kullanarak, Altın ETF fiyatının bir sonraki güne ait değerlerini tahmin etme işlemine geçebilirsiniz.

#Verileri eğitim ve test veri setine ayırın

#Bu adımda, öngörücüleri ve çıktı verilerini eğitim ve test verilerine ayırıyoruz. Eğitim verileri , girdiyi beklenen çıktıyla eşleştirerek doğrusal regresyon modelini oluşturmak için kullanılır.

#Test verileri, modelin ne kadar iyi eğitildiğini tahmin etmek için kullanılır.

#Verilerin ilk %80'i eğitim için, geri kalan veriler ise test için kullanılır.

#X\_train ve y\_train, eğitim veri kümesidir

#X\_test ve y\_test, test veri kümesidir

# Split the data into train and test dataset

t = .8

t = int(t\*len(df))

# Train dataset

X\_train = X[:t]

y\_train = y[:t]

# Test dataset

X\_test = X[t:]

y\_test = y[t:]

Bu kod bloğunda, açıklayıcı değişkenlerinizi ve bağımlı değişkeninizi eğitim ve test veri setlerine ayırıyorsunuz. Bu ayırma işlemi, veri kümenizin ilk %80'ini eğitim veri setine, geri kalan %20'sini de test veri setine atıyor. Bu ayırma işlemini 't' değişkenine atadığınız değere göre yapıyorsunuz.

Daha sonra, 'X\_train' ve 'y\_train' değişkenlerine eğitim veri setini, 'X\_test' ve 'y\_test' değişkenlerine de test veri setini atıyorsunuz. 'X\_train' ve 'y\_train' değişkenleri, eğitim verilerinizi tutacaktır. Bu verileri kullanarak, doğrusal regresyon modeli oluşturabileceksiniz. 'X\_test' ve 'y\_test' değişkenleri ise, modelinizin ne kadar iyi eğitildiğini tahmin etmek için kullanacağınız test verilerini tutacaktır.

#Doğrusal bir regresyon modeli oluşturma

#Şimdi doğrusal bir regresyon modeli oluşturacağız.

#Ancak, doğrusal regresyon nedir?

#'x' ve 'y' değişkenleri arasında, 'y'nin gözlenen değerlerini

#'x'in gözlenen değerleri cinsinden "en iyi" açıklayan matematiksel bir ilişkiyi,

#bir dağılım grafiğinden bir çizgi yerleştirerek yakalamaya çalışırsak,

#o zaman böyle bir denklem x ve y arasındaki doğrusal regresyon analizi denir.

#Daha fazla parçalamak için, regresyon bağımlı bir değişkendeki

#değişimi bağımsız değişkenler açısından açıklar.

#Bağımlı değişken - 'y' tahmin etmek istediğiniz değişkendir.

#Bağımsız değişkenler - 'x', bağımlı değişkeni tahmin etmek için

#kullandığınız açıklayıcı değişkenlerdir. Aşağıdaki regresyon denklemi

#bu ilişkiyi açıklar:

#Ardından, regresyon için katsayı ve sabit oluşturmak üzere

#bağımsız ve bağımlı değişkenleri (x'ler ve y'ler) sığdırmak için

#fit yöntemini kullanırız.

# Doğrusal bir regresyon modeli oluşturma

linear = LinearRegression().fit(X\_train, y\_train)

print("Linear Regression model")

print("Altın Fiyatı (y) = %.2f \* 3 Günlük Hareketli Ortalama (x1) \

+ %.2f \* 9 Günlük Hareketli Ortalama (x2) \

+ %.2f (sabit)" % (linear.coef\_[0], linear.coef\_[1], linear.intercept\_))

Bu kod bloğunda, doğrusal regresyon modeli oluşturuyorsunuz. Doğrusal regresyon, bir bağımlı değişkenin değişimini bağımsız değişkenler açısından açıklar. Örneğin, Altın ETF fiyatının bir sonraki güne ait değerini tahmin etmek istediğinizde, bağımlı değişkeniniz Altın ETF fiyatı olur ve açıklayıcı değişkenleriniz ise, Altın ETF fiyatını tahmin etmek için kullandığınız özelliklerdir.

Doğrusal regresyon modelini oluşturmak için, 'LinearRegression' sınıfını kullanıyorsunuz. Bu sınıfın 'fit()' metodunu kullanarak, eğitim verilerinizi 'X\_train' ve 'y\_train' değişkenlerine sığdırıyorsunuz. 'fit()' metodu, verileri sığdırmak için katsayı ve sabit değerlerini otomatik olarak belirler. Bu katsayı ve sabit değerleri, doğrusal regresyon modelinin denklemini oluşturur.

Son olarak, modelin denklemini ekrana yazdırıyorsunuz. Bu denklem, Altın ETF fiyatının bir sonraki güne ait değerini tahmin etmek için kullanacağınız formüldür. Bu formül, 'x1' ve 'x2' değişkenleri olarak 3 günlük ve 9 günlük hareketli ortalamaları, 'y' değişkeni olarak da Altın ETF fiyatını içerir. Bu formül sayesinde, 3 günlük ve

#Altın ETF fiyatlarını tahmin edin

#Şimdi, modelin test veri setinde çalışıp çalışmadığını kontrol etme zamanı.

#Eğitim veri seti kullanılarak oluşturulan doğrusal modeli kullanarak

#Altın ETF fiyatlarını tahmin ediyoruz. Tahmin yöntemi,

#verilen X açıklayıcı değişkeni için Altın ETF fiyatını (y) bulur.

# Predicting the Gold ETF prices

predicted\_price = linear.predict(X\_test)

predicted\_price = pd.DataFrame(

predicted\_price, index=y\_test.index, columns=['price'])

predicted\_price.plot(figsize=(10, 7))

y\_test.plot()

plt.legend(['Öngörülen\_fiyat', 'Asıl\_fiyat'])

plt.ylabel("Altın Fiyatı")

plt.show()

#Grafik, Gold ETF'nin tahmini ve gerçek fiyatını gösterir.

Bu kod bloğunda, doğrusal regresyon modelini test veri seti üzerinde çalıştırıyorsunuz. Bu işlemi 'predict()' metodu ile yapıyorsunuz. 'predict()' metodu, verilen açıklayıcı değişkenleri kullanarak, bağımlı değişkenin (Altın ETF fiyatının bir sonraki güne ait değerini) tahmin eder.

Daha sonra, tahmin edilen fiyatları 'predicted\_price' değişkenine atıyorsunuz ve bu değişkeni bir Pandas DataFrame'ine dönüştürüyorsunuz. Bu DataFrame, tahmin edilen fiyatların tarihlere göre sıralanmış halini gösterir.

Son olarak, tahmin edilen fiyatları ve gerçek fiyatları bir grafikte gösteriyorsunuz. Bu grafikte, mavi çizgi olarak tahmin edilen fiyatları, kırmızı çizgi olarak da gerçek fiyatları gö

#score() işlevini kullanarak uyumun iyiliğini hesaplayalım.

# R square

r2\_score = linear.score(X[t:], y[t:])\*100

float("{0:.2f}".format(r2\_score))

#Görüldüğü gibi modelin R-kare değeri %99,35'dir.

#R-kare her zaman %0 ile %100 arasındadır.

#%100'e yakın bir puan, modelin Altın ETF fiyatlarını iyi açıkladığını gösterir.

Bu kod bloğunda, doğrusal regresyon modelinin uyumunun iyiliğini ölçüyorsunuz. Bu ölçümü, 'score()' metodu ile yapıyorsunuz. 'score()' metodu, verilen açıklayıcı değişkenleri ve bağımlı değişkeni kullanarak, modelin doğruluğunu ölçer.

Bu ölçüm için kullandığınız metrik, R-kare değeridir. R-kare, bir modelin bağımlı değişkenin değişimini, açıklayıcı değişkenler açısından ne kadar iyi açıkladığını ölçer. R-kare değeri, her zaman 0 ile 100 arasında olur ve 100'e yakın bir puan, modelin bağımlı değişkeni iyi açıkladığını gösterir.

Bu ölçümde, R-kare değerinizin %99,35 olduğunu görüyorsunuz. Bu değer, modelin Altın ETF f

#Kümülatif getirileri çizme

#Performansını analiz etmek için bu stratejinin kümülatif getirilerini

#hesaplayalım.

#1-Kümülatif getirileri hesaplama adımları aşağıdaki gibidir:

#2-Altın fiyatının günlük yüzde değişimini oluştur

#3-Bir sonraki günün tahmini fiyatı mevcut günün tahmini fiyatından

#fazla olduğunda "1" ile temsil edilen bir alım satım sinyali oluşturun.

#Aksi halde pozisyon alınmaz

#4-Günlük yüzde değişimini ticaret sinyaliyle çarparak strateji getirilerini

#hesaplayın.

#5-Son olarak, kümülatif getiri grafiğini çizeceğiz

gold = pd.DataFrame()

gold['price'] = df[t:]['Close']

gold['predicted\_price\_next\_day'] = predicted\_price

gold['actual\_price\_next\_day'] = y\_test

gold['gold\_returns'] = gold['price'].pct\_change().shift(-1)

gold['signal'] = np.where(gold.predicted\_price\_next\_day.shift(1) < gold.predicted\_price\_next\_day,1,0)

gold['strategy\_returns'] = gold.signal \* gold['gold\_returns']

((gold['strategy\_returns']+1).cumprod()).plot(figsize=(10,7),color='g')

plt.ylabel('Cumulative Returns')

plt.show()

Bu kod bloğunda, doğrusal regresyon modelini kullanarak Altın ETF'nin gelecekteki fiyatlarını tahmin etmeyi deneyen bir stratejinin performansını ölçüyorsunuz. Bu işlemi, stratejinin oluşturduğu kümülatif getirileri gösteren bir grafik çizerek yapıyorsunuz.

İlk olarak, 'gold' isimli bir Pandas DataFrame'i oluşturuyorsunuz. Bu DataFrame'e, Altın ETF'nin günlük fiyatlarını ve tahmin edilen fiyatlarını içeren 'price' ve 'predicted\_price\_next\_day' sütunlarını ekliyorsunuz. Daha sonra, 'actual\_price\_next\_day' sütununa da gerçek fiyatları ekliyorsunuz.

Ardından, 'gold\_returns' sütununa, Altın ETF'nin günlük yüzde değişimlerini içeren verileri ekliyorsunuz. Bu değişimler, 'pct\_change()' metodu ile hesaplanıyor ve 'shift(-1)' metodu ile bir gün geri kaydırılıyor. Bu işlem, bir sonraki günün fiyatının bugünkü fiyatına oranını gösterir.

Son olarak

#Sharpe oranı, Nobel ödüllü William F. Sharpe tarafından geliştirilmiştir

#ve yatırımcıların bir yatırımın riskine kıyasla getirisini anlamalarına

#yardımcı olmak için kullanılır. Oran, birim oynaklık veya toplam risk başına

#risksiz oranı aşan ortalama getiridir. Oynaklık, bir varlık veya portföyün

#fiyat dalgalanmalarının bir ölçüsüdür

#Sharpe oranını hesaplama formülü {R (p) – R (f)} /s (p)’dir.

#Burada

#R (p): Portföy getirisi

#R (f): Risksiz getiri oranı

#s (p): Portföy standart sapması

sharpe = gold['strategy\_returns'].mean()/gold['strategy\_returns'].std()\*(252\*\*0.5)

'Sharpe Oranı %.2f' % (sharpe)

Bu kod bloğunda, 'gold' DataFrame'i üzerinden bir stratejinin Sharpe oranını hesaplıyorsunuz. Sharpe oranı, bir yatırımın riskine kıyasla getirisini ölçen bir metriktir. Bu oran, birim oynaklık veya toplam risk başına risksiz oranı aşan ortalama getiridir.

Sharpe oranını hesaplamak için, strateji getirilerinin ortalamasını strateji getirilerinin standart sapmasına bölüyorsunuz. Böylece, bir yatırımın getirisi ile oynaklık arasındaki ilişkiyi ölçmüş olursunuz. Daha sonra, bu oranı 252 günlük bir yıl olarak kabul ederek karekök alıyorsunuz. Bu sayı, yıl içinde tahvil faiz oranlarının üstünde bir getiri elde etmek için kaç gün süreçte bir yatırım yapılması gerektiğini gösterir.

Son olarak, hesa

#The below output is based on the data points in the train dataset.

#Sometimes, it is difficult to understand why there is more weight to 3 days moving average compared to 9 days moving average.

#But this equation is created by what the machine learning model learnt from the data.

#Gold ETF Price = 1.2 \* 3 Days Moving Average – 0.2 \* 9 Days Moving Average + 0.39

#Using this equation, you can predict the next day price of the Gold ETF and buy/sell if the next day price is significantly

#higher or lower than the previous day price.

#To learn more about R2 score you can refer to the following link

#Günlük hareketleri tahmin etmek için bu model nasıl kullanılır?

#Altın fiyatlarını tahmin etmek ve GLD almamız mı yoksa

#pozisyon almamamız mı gerektiğine dair bir ticaret sinyali vermek için

#aşağıdaki kodu kullanabilirsiniz.

#import datetime and get today's date

# Get the data rolling hareketli ortalama almak için kullanılır

import datetime as dt

current\_date = dt.datetime.now()

# Get the data

df['S\_3'] = df['Close'].rolling(window=3).mean()

df['S\_9'] = df['Close'].rolling(window=9).mean()

df = df.dropna()

# Forecast the price

df['predicted\_gold\_price'] = linear.predict(df[['S\_3', 'S\_9']])

df['signal'] = np.where(df.predicted\_gold\_price.shift(1) < df.predicted\_gold\_price,"Buy","No Position")

# Print the forecast

df.tail(1)[['signal','predicted\_gold\_price']].T

Açıklayıcı değişkenler, bir değişkenin değerini açıklamaya çalıştığımız diğer değişkenlerdir. Örneğin, Altın ETF fiyatını tahmin etmek için kullanmak istediğimiz özelliklerdir. Bu stratejide açıklayıcı değişkenler, son 3 gün ve 9 günün hareketli ortalamalarıdır. Bağımlı değişken ise, açıklayıcı değişkenlerin değerlerine bağlı olarak tahmin etmeye çalıştığımız değişkendir. Örneğin, tahmin etmeye çalıştığımız Altın ETF fiyatıdır.

Verileri eğitim ve test veri setine ayırmak, öngörücüleri ve çıktı verilerini eğitim ve test verilerine ayırmak için kullanılır. Eğitim verileri, girdiyi beklenen çıktıyla eşleştirerek doğrusal regresyon modelini oluşturmak için kullanılır. Test verileri ise, modelin ne kadar iyi eğitildiğini tahmin etmek için kullanılır. Verilerin ilk yüzde sekseni eğitim için, geri kalan veriler ise test için

from statsmodels.tsa.api import ARIMA

model = ARIMA(df['Close'],order =(0,0,0))

result = model.fit()

print(result.aic)

AIC (Akaike İnformation Criterion) bir modele uygunluk ölçümü olarak kullanılır. Modelin uygunluğunu ölçmek için kullanılan bir kriterdir. AIC değeri düşük olan model daha uygun bir model olarak kabul edilir. Bu nedenle AIC değeri en düşük olan model seçilir. Bu örnekte, ARIMA modeli için AIC değeri hesaplanmıştır.

model2 = ARIMA(df['Close'], order=(0,1,0))

result2 = model2.fit()

print(result2.aic)

Akaike bilgi kriterine göre (AIC), verilen bir veri kümesi için istatistiksel modellerin göreli kalitesini ölçer. Sıklıkla, modeller arasındaki AIC değerlerini karşılaştırarak en iyi modeli seçmek için kullanılır. Bu durumda, df veri çerçevesinin "Close" sütununa iki farklı ARIMA modeli uydurmaya çalışıyorsunuz. İlk modelin ARIMA sırası (0,0,0) iken, ikinci modelin sırası (0,1,0)'dir.

İlk modelin AIC değeri, ilk model uydurma sonucu olarak yazdırılır. İkinci modelin AIC değeri ise, ikinci model uydurma sonucu olarak yazdırılır.

Genel olarak, daha düşük bir AIC değeri, daha iyi bir model uydurma anlamına gelir. Bu nedenle, eğer ikinci modelin ilk modelden daha düşük bir AIC değeri varsa, veri için daha iyi bir uydurma olarak kabul edilebilir.

model3 = ARIMA(df['Close'], order = (1,0,0))

result3 = model3.fit()

print(result3.aic)

Akaike bilgi kriterine (AIC) bir veri kümesi için istatistiksel modellerin göreli kalitesi ölçülür. AIC değerlerini karşılaştırarak, modeller arasından en iyi modeli seçmeyi amaçlar. Bu durumda, "Close" sütununu içeren df veri çerçevesine (1,0,0) sırasıyla bir ARIMA modeli uyduruyorsunuz. Bu modelin AIC değeri, model uydurma sonucu olarak yazdırılır.

Genel olarak, düşük AIC değeri daha iyi bir model uydurmayı gösterir. Bu nedenle, eğer bu model önceki iki modelden daha düşük bir AIC değeri varsa, verilere için daha iyi bir uydurma olur.

model4 = ARIMA(df['Close'], order =(1,1,0))

result4 = model4.fit()

print(result4.aic)

Bu kod parçacığı, veri kümesi 'Close' değişkenini tahmin etmek için ARIMA modeli kullanır. ARIMA (AutoRegressive Integrated Moving Average) modeli, bir zaman serisi verisi üzerinde düzenleyici bir tahmin modelidir. Bu model, bir zaman serisi verisi için tahmin yapmak için geçmiş verilere dayanır ve bu nedenle "auto-regresif" olarak adlandırılır. Bu modelin sıralaması (1,1,0) olarak belirtilmiştir. Bu, modelin otomatik-regresif bileşenlerinin birinci dereceden ve hareketli ortalama bileşenlerinin sıfırıncı dereceden olduğunu gösterir.

model4 değişkenine atanan ARIMA modelinin fit() metodu çağrılır ve sonuç result4 değişkenine atanır. Daha sonra, result4.aic (Akaike İyileştirme İndeksi) öğesi yazdırılır. Akaike İyileştirme İndeksi, bir modelin veri kümesine uygunluğunu ölçmek için kullanılan bir ölçümdür. Bu ölçüm, daha düşük değerler modelin daha iyi uyum sağladığını gösterir.

model5 = ARIMA(df['Close'], order = (1,1,1))

result5 = model5.fit()

print(result5.aic)

Bu kod parçacığı, veri kümesi 'Close' değişkenini tahmin etmek için ARIMA modeli kullanır. ARIMA (AutoRegressive Integrated Moving Average) modeli, bir zaman serisi verisi üzerinde düzenleyici bir tahmin modelidir. Bu model, bir zaman serisi verisi için tahmin yapmak için geçmiş verilere dayanır ve bu nedenle "auto-regresif" olarak adlandırılır. Bu modelin sıralaması (1,1,1) olarak belirtilmiştir. Bu, modelin otomatik-regresif bileşenlerinin birinci dereceden, hareketli ortalama bileşenlerinin birinci dereceden ve gözlemlenen serinin birinci dereceden türevini içerdiğini gösterir.

model5 değişkenine atanan ARIMA modelinin fit() metodu çağrılır ve sonuç result5 değişkenine atanır. Daha sonra, result5.aic (Akaike İyileştirme İndeksi) öğesi yazdırılır. Akaike İyileştirme İndeksi, bir modelin veri kümesine uygunluğunu ölçmek için kullanılan bir ölçümdür. Bu ölçüm, daha düşük değerler modelin daha iyi uyum sağladığını gösterir.

Bu iki kod parçacığı arasındaki fark, kullanılan ARIMA modelinin sıralamasıdır. Model4 için sıralama (1,1,0) kullanılırken, model5 için sıralama (1,1,1) kullanılmıştır. Bu sıralama, modelin hangi dereceden otomatik-regresif bileşenleri, hangi dereceden hareketli ortalama bileşenleri ve hangi dereceden türevleri içerdiğini belirtir. Bu sıralamanın değişimi, modelin tahmin gücünü etkileyebilir ve bu nedenle, hangisinin daha iyi uyum sağladığını belirlemek için Akaike İyileştirme İndeksi kullanılır.

residuals = pd.DataFrame(result5.resid)

residuals = residuals.iloc[1:]

residuals.describe()

Bu kod parçacığı, result5 modelinin residuals öğesini (artıklar) kullanarak bir veri kümesi oluşturur. Residuals, modelin tahminlerini gerçek değerlerden çıkartıp elde edilen farktır. Eğer model doğru bir şekilde tahmin etmişse, residuals değerleri normalde dağılımı olan bir rasgele değişken gibi görünecektir.

Residuals veri kümesi, iloc[] metodu kullanılarak indeks numaralarına göre veri seçimi yapılarak sıfırdan birinci satırı atlar. Daha sonra, residuals veri kümesinin describe() metodu çağrılır ve veri kümesinin özet istatistikleri (ortalama, medyan, standart sapması, vb.) gösterilir. Bu istatistikler, residuals değerlerinin dağılımı hakkında bilgi verir ve modelin tahminlerinin ne kadar doğru olduğunu ölçmek için kullanılabilir.

plt.style.use('fivethirtyeight')

plt.figure(figsize = (18,10))

plt.plot(df['Open'])

plt.plot(df['Close'])

plt.legend(['Open','Close'])

plt.title('Open and Close Prices')

Bu kod parçacığı, veri kümesinde 'Open' ve 'Close' değişkenlerinin değerlerini içeren iki grafik oluşturur. Öncelikle, plt.style.use() fonksiyonu kullanılarak 'fivethirtyeight' stili ayarlanır. Daha sonra, plt.figure() fonksiyonu kullanılarak grafiklerin ölçüleri belirlenir. Ölçüler (18,10) olarak belirtilmiştir.

Sonra, plt.plot() fonksiyonu kullanılarak 'Open' ve 'Close' değişkenlerinin değerleri grafiklerde gösterilir. plt.legend() fonksiyonu kullanılarak grafiklerin lejantları ('Open' ve 'Close') eklenir ve plt.title() fonksiyonu kullanılarak grafiklerin başlıkları ('Open and Close Prices') belirlenir. Bu kod parçacığı sonucunda, 'Open' ve 'Close' değişkenlerinin değerlerini gösteren iki grafik çıktısı alınır.

plt.figure(figsize = (18,10))

plt.plot(df['High'])

plt.plot(df['Low'])

plt.legend(['High','Low'])

plt.title('High and Low Prices of Gold', fontsize =20)

Bu kod parçacığı, veri kümesinde 'High' ve 'Low' değişkenlerinin değerlerini içeren iki grafik oluşturur. Öncelikle, plt.figure() fonksiyonu kullanılarak grafiklerin ölçüleri belirlenir. Ölçüler (18,10) olarak belirtilmiştir.

Sonra, plt.plot() fonksiyonu kullanılarak 'High' ve 'Low' değişkenlerinin değerleri grafiklerde gösterilir. plt.legend() fonksiyonu kullanılarak grafiklerin lejantları ('High' ve 'Low') eklenir ve plt.title() fonksiyonu kullanılarak grafiklerin başlıkları ('High and Low Prices of Gold') belirlenir. Bu kod parçacığı sonucunda, 'High' ve 'Low' değişkenlerinin değerlerini gösteren iki grafik çıktısı alınır. Başlık font büyüklüğü 20 olarak belirtilmiştir.

plt.plot(df['Close'])

plt.plot(residuals)

plt.legend(['Close','Residuals'])

Bu kod parçacığı, veri kümesinde 'Close' değişkeninin değerlerini ve residuals (artıklar) veri kümesinin değerlerini içeren iki grafik oluşturur. Öncelikle, plt.plot() fonksiyonu kullanılarak 'Close' değişkeninin değerleri ve residuals veri kümesinin değerleri grafiklerde gösterilir. Daha sonra, plt.legend() fonksiyonu kullanılarak grafiklerin lejantları ('Close' ve 'Residuals') eklenir.

Bu kod parçacığı, 'Close' değişkeninin tahmin edilen değerlerini ve gerçek değerlerden fark olarak elde edilen residuals değerlerini gösterir. Bu grafik, modelin tahminlerinin ne kadar doğru olduğunu gösterir. Eğer model doğru bir şekilde tahmin etmişse, residuals değerleri normalde dağılımı olan bir rasgele değişken gibi görünecektir.

df['PriceDiff'] = df['Close'].shift(-1) - df['Close']

df = df.dropna()

print(df['PriceDiff'])

Bu kod parçacığı, veri kümesine 'PriceDiff' adında bir yeni değişken ekler. Bu değişken, 'Close' değişkeninin bir sonraki değeri ile 'Close' değişkeninin şuanki değerinin farkını gösterir. Örneğin, df['Close'][0] değeri ile df['Close'][1] değeri arasındaki fark df['PriceDiff'][0] değerine atanır.

df['Close'].shift(-1) fonksiyonu, 'Close' değişkeninin değerlerini bir satır aşağı kaydırarak yeni bir veri kümesi oluşturur. Bu veri kümesi, 'Close' değişkeninin değerlerinin bir sonraki değerlerini gösterir. Bu veri kümesi ile 'Close' değişkeninin şuanki değerleri arasındaki farkı alıp 'PriceDiff' değişkenine atar.

Sonra, df.dropna() fonksiyonu kullanılarak veri kümesinde eksik (NaN) değerler atılır. Bu işlemden sonra, df['PriceDiff'] değişkeninin değerleri yazdırılır.

Bu kod parçacığı, 'Close' değişkeninin bir sonraki değerlerinin ne kadar farklı olduğunu gösterir. Bu değerlerin ortalaması, 'Close' değişkeninin bir sonraki değerlerinin ne kadar farklı olacağını tahmin etmek için kullanılabilir.

plt.figure(figsize = (18,10))

plt.plot(df['PriceDiff'])

plt.plot(df['Close'])

plt.legend(['PriceDiff','Close'])

plt.title('Price Difference and Close Price of Gold',fontsize = 30)

Bu kod parçacığı, veri kümesinde 'PriceDiff' ve 'Close' değişkenlerinin değerlerini içeren iki grafik oluşturur. Öncelikle, plt.figure() fonksiyonu kullanılarak grafiklerin ölçüleri belirlenir. Ölçüler (18,10) olarak belirtilmiştir.

Sonra, plt.plot() fonksiyonu kullanılarak 'PriceDiff' ve 'Close' değişkenlerinin değerleri grafiklerde gösterilir. Daha sonra, plt.legend() fonksiyonu kullanılarak grafiklerin lejantları ('PriceDiff' ve 'Close') eklenir ve plt.title() fonksiyonu kullanılarak grafiklerin başlıkları ('Price Difference and Close Price of Gold') belirlenir. Başlık font büyüklüğü 30 olarak belirtilmiştir.

Bu kod parçacığı, 'PriceDiff' değişkeninin 'Close' değişkenine olan etkisini gösterir. 'PriceDiff' değişkeninin değerlerinin 'Close' değişkeninin değerlerine nasıl uyum sağladığını gösterir. Bu grafik, 'Close' değişkeninin bir sonraki değerlerinin ne kadar farklı olacağını tahmin etmek için kullanılabilir.

df['MA10'] = df['Close'].rolling(10).mean()

df['MA50'] = df['Close'].rolling(50).mean()

Bu kod parçacığı, veri kümesine 'MA10' ve 'MA50' adında iki yeni değişken ekler. 'MA10' değişkeni, 'Close' değişkeninin 10 günlük hareketli ortalamasını gösterir. 'MA50' değişkeni ise, 'Close' değişkeninin 50 günlük hareketli ortalamasını gösterir.

df['Close'].rolling() fonksiyonu, veri kümesinin belirtilen periyot için hareketli ortalamasını hesaplar. Örneğin, df['Close'].rolling(10).mean() fonksiyonu, 'Close' değişkeninin 10 günlük hareketli ortalamasını hesaplar. Bu fonksiyon sonucu, 'MA10' değişkenine atanır. Benzer şekilde, df['Close'].rolling(50).mean() fonksiyonu 'Close' değişkeninin 50 günlük hareketli ortalamasını hesaplar ve 'MA50' değişkenine atanır.

Bu kod parçacığı, 'Close' değişkeninin değişimlerinin nasıl bir trend izlediğini gösterir. Hareketli ortalamalar, veri kümesindeki değişkenlerin değişimlerinin düzenli bir şekilde ortalamasını gösterir ve bu nedenle trendlerin belirlenmesinde kullanılabilir.

from statsmodels.tsa.stattools import adfuller

def adf\_test(timeseries):

#Perform Dickey-Fuller test:

print ('Results of Dickey-Fuller Test:')

dftest = adfuller(timeseries, autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

for key,value in dftest[4].items():

dfoutput['Critical Value (%s)'%key] = value

print (dfoutput)

Bu kod parçacığı, Dickey-Fuller testini gerçekleştirmeye yarayan adf\_test() fonksiyonunu tanımlar. Bu fonksiyon, bir zaman serisi veri kümesi alır ve Dickey-Fuller testini uygular.

Dickey-Fuller testi, bir zaman serisinin durağan olup olmadığını test etmek için kullanılır. Bir zaman serisi durağandır, eğer ortalaması ve varyansı zamanla değişmezse. Eğer bir zaman serisi durağan değilse, bu zaman serisi trende sahiptir ve bu nedenle modelleme yapılması zorlaşır.

Dickey-Fuller testi, 'Test Statistic' değeri, 'p-value' değeri ve 'Critical Value' değerleri gibi bazı önemli istatistikleri içeren bir çıktı verir. 'Test Statistic' değeri, null hipotezinin reddedilip reddedilmeyeceğini gösterir. Eğer 'Test Statistic' değeri 'Critical Value' değerlerinden düşükse, null hipotezi reddedilir ve zaman serisi durağandır. Eğer 'Test Statistic' değeri 'Critical Value' değerlerinden yüksekse, null hipotezi kabul edilir ve zaman serisi durağan değildir. 'p-value' değeri ise, null hipotezinin reddedilme olasılığını gösterir. Eğer 'p-value' değeri düşükse, null hipotezi daha güçlü bir şekilde reddedilir.

print(adf\_test(df['High']))

Bu kod parçacığı, adf\_test() fonksiyonunu kullanarak Dickey-Fuller testini 'High' değişkeni için gerçekleştirir. Bu fonksiyon, 'High' değişkeninin durağan olup olmadığını test eder.

Dickey-Fuller testi çıktısı, 'Test Statistic' değeri, 'p-value' değeri, '#Lags Used' değeri, 'Number of Observations Used' değeri ve 'Critical Value' değerlerini içerir. Bu değerler, 'High' değişkeninin durağan olup olmadığı hakkında bilgi verir. 'Test Statistic' değeri, 'Critical Value' değerlerinden düşükse 'High' değişkeni durağandır. 'p-value' değeri ise, null hipotezinin reddedilme olasılığını gösterir. Eğer 'p-value' değeri düşükse, null hipotezi daha güçlü bir şekilde reddedilir ve 'High' değişkeni durağandır.

from statsmodels.tsa.stattools import kpss

def kpss\_test(timeseries):

print("Results of KPSS Test:")

kpsstest = kpss(timeseries, regression="c", nlags="auto")

kpss\_output = pd.Series(

kpsstest[0:3], index=["Test Statistic", "p-value", "Lags Used"]

)

for key, value in kpsstest[3].items():

kpss\_output["Critical Value (%s)" % key] = value

print(kpss\_output)

Bu kod parçacığı, KPSS (Kwiatkowski-Phillips-Schmidt-Shin) testini gerçekleştirmeye yarayan kpss\_test() fonksiyonunu tanımlar. Bu fonksiyon, bir zaman serisi veri kümesi alır ve KPSS testini uygular.

KPSS testi, bir zaman serisinin durağan olup olmadığını test etmek için kullanılır. Bir zaman serisi durağandır, eğer ortalaması ve varyansı zamanla değişmezse. Eğer bir zaman serisi durağan değilse, bu zaman serisi trende sahiptir ve bu nedenle modelleme yapılması zorlaşır.

KPSS testi, 'Test Statistic' değeri, 'p-value' değeri ve 'Critical Value' değerleri gibi bazı önemli istatistikleri içeren bir çıktı verir. 'Test Statistic' değeri, null hipotezinin reddedilip reddedilmeyeceğini gösterir. Eğer 'Test Statistic' değeri 'Critical Value' değerlerinden yüksekse, null

kpss\_test(df['High'])

Bu kod parçacığı, kpss\_test() fonksiyonunu kullanarak KPSS testini 'High' değişkeni için gerçekleştirir. Bu fonksiyon, 'High' değişkeninin durağan olup olmadığını test eder.

KPSS testi çıktısı, 'Test Statistic' değeri, 'p-value' değeri ve 'Critical Value' değerlerini içerir. Bu değerler, 'High' değişkeninin durağan olup olmadığı hakkında bilgi verir. 'Test Statistic' değeri, 'Critical Value' değerlerinden yüksekse 'High' değişkeni durağandır. 'p-value' değeri ise, null hipotezinin reddedilme olasılığını gösterir. Eğer 'p-value' değeri düşükse, null hipotezi daha güçlü bir şekilde reddedilir ve 'High' değişkeni durağandır.

df['High\_diff'] = df['High'] - df['High'].shift(1)

df['High\_diff'].dropna().plot(figsize = (20,10))

Bu kod parçacığı, 'High' değişkeninin bir önceki değerlerine göre farkını hesaplayan 'High\_diff' adlı yeni bir değişken oluşturur. Öncelikle, 'High\_diff' değişkenine 'High' değişkeninin bir önceki değerlerine göre farkı atanır. df['High'].shift(1) fonksiyonu, veri kümesindeki 'High' değişkeninin bir önceki değerlerini döndürür. Bu fonksiyon sonucu, 'High' değişkeninden çıkarılarak 'High\_diff' değişkenine atanır.

Daha sonra, df['High\_diff'].dropna() fonksiyonu ile 'High\_diff' değişkenindeki verilerin NaN değerleri çıkarılır. Son olarak, plt.plot() fonksiyonu kullanılarak 'High\_diff' değişkeninin değerleri grafikte gösterilir. Grafik ölçüleri (20,10) olarak belirlenir.

Bu kod parçacığı, 'High' değişkeninin değerlerinin bir önceki değerlerine göre nasıl değiştiğini gösterir. Bu grafik, 'High' değişkeninin değişimlerinin trendlerini gösterir ve bu nedenle trendlerin belirlenmesinde kullanılabilir.

kpss\_test(df['High\_diff'].dropna())

Bu kod parçacığı, kpss\_test() fonksiyonunu kullanarak KPSS testini 'High\_diff' değişkeni için gerçekleştirir. Bu fonksiyon, 'High\_diff' değişkeninin durağan olup olmadığını test eder.

KPSS testi çıktısı, 'Test Statistic' değeri, 'p-value' değeri ve 'Critical Value' değerlerini içerir. Bu değerler, 'High\_diff' değişkeninin durağan olup olmadığı hakkında bilgi verir. 'Test Statistic' değeri, 'Critical Value' değerlerinden yüksekse 'High\_diff' değişkeni durağandır. 'p-value' değeri ise, null hipotezinin reddedilme olasılığını gösterir. Eğer 'p-value' değeri düşükse, null hipotezi daha güçlü bir şekilde reddedilir ve 'High\_diff' değişkeni durağandı

adf\_test(df['High\_diff'].dropna())

Results of Dickey-Fuller Test:

Test Statistic -1.627129e+01

p-value 3.485839e-29

#Lags Used 2.500000e+01

Number of Observations Used 5.675000e+03

Critical Value (1%) -3.431503e+00

Critical Value (5%) -2.862049e+00

Critical Value (10%) -2.567041e+00

dtype: float64

Bu çıktı, Dickey-Fuller testi sonuçlarını gösterir. Bu test, 'High\_diff' değişkeninin durağan olup olmadığını test etmiştir.

'Test Statistic' değeri, null hipotezinin reddedilip reddedilmeyeceğini gösterir. Eğer 'Test Statistic' değeri 'Critical Value' değerlerinden düşükse, null hipotezi reddedilir ve 'High\_diff' değişkeni durağandır. Bu çıktıda görüldüğü gibi, 'Test Statistic' değeri 'Critical Value' değerlerinden düşüktür ve bu nedenle 'High\_diff' değişkeni durağandır.

'p-value' değeri ise, null hipotezinin reddedilme olasılığını gösterir. Eğer 'p-value' değeri düşükse, null hipotezi daha güçlü bir şekilde reddedilir. Bu çıktıda görüldüğü gibi,

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

import math

from keras.layers import Dense, LSTM

import matplotlib.pyplot as plt

plt.style.use("fivethirtyeight")

#Create a new dataframe with only the "Close column"

data = df.filter(['Close'])

#Convert the dataframe to a numpy array

dataset = data.values

#Get the number of rows to train the model on

training\_data\_len = math.ceil(len(dataset)\* .8)

training\_data\_len

Bu kod parçacığı, bir zaman serisi veri kümesinde bir LSTM (Long Short-Term Memory) yapay sinir ağı modeli eğitmek için gerekli adımları içerir.

Öncelikle, veri kümesinden 'Close' değişkeni kullanılarak yeni bir veri kümesi oluşturulur. Bu veri kümesi, data adlı bir değişkene atanır. Daha sonra, data değişkeni numpy dizisine dönüştürülür ve dataset değişkene atanır.

Son olarak, dataset veri kümesinin %80'i kadarının model eğitim verisi olarak kullanılacağını belirten training\_data\_len değişkeni hesaplanır. Bu değişken, veri kümesinin satır sayısının yuvarlanması ile hesaplanır. Bu adım, model eğitiminde kullanılacak verinin kaç satırdan oluşacağını belirtir.

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(dataset)

scaled\_data

array([[0.01494925],

[0.01406664],

[0.01423213],

...,

[0.8162511 ],

[0.81067961],

[0.80135702]])

Bu kod parçacığı, MinMaxScaler sınıfını kullanarak dataset veri kümesinin ölçeklendirilmesini gerçekleştirir. Ölçeklendirme, veri kümesindeki değerlerin aynı aralıkta olmasını sağlar ve bu sayede modelin daha iyi çalışmasını sağlar.

MinMaxScaler sınıfı, veri kümesinin değerlerini 0 ile 1 arasında bir değer aralığına dönüştürür. Bu sınıf, 'feature\_range' parametresi ile dönüştürülecek değer aralığını belirtir. Ölçeklendirme işlemi, scaled\_data değişkenine atanarak gerçekleştirilir.

scaled\_data değişkeni, dataset veri kümesinin ölçeklendirilmiş halini içerir. Ölçeklendirme işleminden sonra veri kümesindeki değerler, 0 ile 1 arasında bir değer aralığında olacaktır

#create the training data set

#Create the scaled training data set

train\_data = scaled\_data[0:training\_data\_len , :]

#Split the data into x\_train and y\_train data sets

x\_train = []

y\_train = []

for i in range(60, len(train\_data)):

x\_train.append(train\_data[i-60:i, 0])

y\_train.append(train\_data[i, 0])

if i<= 60:

print(x\_train)

print(y\_train)

print()

Bu kod parçacığı, ölçeklendirilmiş veri kümesinden eğitim verisi oluşturur. Ölçeklendirilmiş veri kümesinin %80'i kadarı, train\_data değişkenine atanır ve bu değişken eğitim verisi olarak kullanılır.

Daha sonra, x\_train ve y\_train adlı iki veri kümesi oluşturulur. Bu veri kümelerinin oluşturulma işlemi for döngüsü kullanılarak gerçekleştirilir. x\_train veri kümesine, train\_data veri kümesinin i-60 ile i arasındaki değerleri eklenir. y\_train veri kümesine ise, train\_data veri kümesinin i. satırındaki değer eklenir. Bu işlem, train\_data veri kümesinin tüm satırları için tekrar edilir.

Bu kod parçacığı, x\_train ve y\_train veri kümelerinin oluşturulmasını gerçekleştirir. Bu veri kümelerinin oluşturulma işlemi, LSTM yapay sinir ağı modelinin eğitiminde kullanılacak veri kümelerinin oluşturulması için gereklidir.

#Convert the x\_train and y\_train to numpy arrays

x\_train, y\_train = np.array(x\_train), np.array(y\_train)

Bu kod parçacığı, x\_train ve y\_train veri kümelerini numpy dizilerine dönüştürür. LSTM yapay sinir ağı modelinin eğitiminde kullanılacak veri kümelerinin numpy dizisi olarak olması gerekir. Bu nedenle, x\_train ve y\_train veri kümeleri np.array() fonksiyonu kullanılarak numpy dizisine dönüştürülür.

Bu işlem sonrasında, x\_train ve y\_train veri kümeleri numpy dizisi olarak kullanılabilir hale gelir. Bu veri kümelerinin numpy dizisi olması, LSTM yapay sinir ağı modelinin eğitimi için gereklidir.

#Reshape the data

x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

x\_train.shape

Bu kod parçacığı, x\_train veri kümesinin şeklini değiştirir. LSTM yapay sinir ağı modelinin eğitiminde, veri kümesinin 3 boyutlu olması gerekir. Bu nedenle, x\_train veri kümesi np.reshape() fonksiyonu kullanılarak 3 boyutlu hale getirilir.

np.reshape() fonksiyonu, veri kümesinin şeklini değiştirerek, veri kümesini istenen şekle dönüştürür. Bu fonksiyon, veri kümesinin yeni şeklini belirten iki parametre alır. İlk parametre, veri kümesinin yeni satır sayısını, ikinci parametre ise veri kümesinin yeni sütun sayısını belirtir.

Bu kod parçacığında, x\_train veri kümesi (x\_train.shape[0], x\_train.shape[1], 1) şekline dönüştürülür. Bu şekil, veri kümesinin satır sayısının x\_train.shape[0] kadar olduğunu, sütun sayısının da x\_train.shape[1] kadar olduğunu belirtir. Bu işlem sonrasında, x\_train veri kümesi 3 boyutlu hale gelir ve LSTM yapay sinir ağı modelinin eğitimi için hazır hale gelir.

#Build the LSTM model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape= (x\_train.shape[1], 1)))

model.add(LSTM(50, return\_sequences= False))

model.add(Dense(25))

model.add(Dense(1))

Bu kod parçacığında, LSTM yapay sinir ağı modeli oluşturulur. Bu model, Sequential() sınıfı kullanılarak oluşturulur. Sequential() sınıfı, yapay sinir ağı modelinin karmaşık bir yapısı olmadığı durumlarda kullanılır ve tek yönlü bir yapıya sahiptir.

Daha sonra, model nesnesine LSTM katmanları eklenir. İlk LSTM katmanı, 50 tane çıktı üretecek şekilde oluşturulur ve return\_sequences parametresi True olarak belirtilir. Bu parametre, LSTM katmanının çıktısının diğer katmanlara iletilip iletilmeyeceğini belirtir. İkinci LSTM katmanı ise, 50 tane çıktı üretecek şekilde oluşturulur ve return\_sequences parametresi False olarak belirtilir. Bu parametre, ikinci LSTM katmanının çıktısının diğer katmanlara iletilmeyeceğini belirtir.

Son olarak, model nesnesine Dense katmanları eklenir. İlk Dense katmanı, 25 tane çıktı üretecek şekilde oluşturulur. İkinci Dense katmanı ise, 1 tane çıktı üretecek şekilde oluşturulur. Bu katman, yapay sinir ağı modelinin çıktı katmanıdır.

Bu kod parçacığında, LSTM yapay sinir ağı modelinin oluşturulması gerçekleştirilir. Bu model, veri kümesini işleyerek tahminler yapacak şekilde oluşturulur.

#Compile the model

model.compile(optimizer = 'adam', loss='mean\_squared\_error')

Bu kod parçacığında, oluşturulan LSTM yapay sinir ağı modeli derlenir. model.compile() fonksiyonu, yapay sinir ağı modelinin derlenmesi için kullanılır. Bu fonksiyon, modelin eğitim sırasında kullanılacak optimizasyon algoritmasını ve kayıp fonksiyonunu belirler.

Bu kod parçacığında, optimizer parametresine 'adam' değeri ve loss parametresine 'mean\_squared\_error' değeri verilir. Bu parametreler, yapay sinir ağı modelinin Adam optimizasyon algoritmasını kullanarak eğitileceğini ve kayıp fonksiyonunun kare ortalama hatası olduğunu belirtir.

Bu işlem sonrasında, LSTM yapay sinir ağı modeli derlenmiş olur ve eğitim işlemine hazır hale gelir.

#Train the model

model.fit(x\_train, y\_train, batch\_size=1, epochs=1)

Bu kod parçacığında, LSTM yapay sinir ağı modeli eğitilir. model.fit() fonksiyonu, yapay sinir ağı modelinin eğitimini gerçekleştirir. Bu fonksiyon, eğitim verilerini alır ve modelin eğitimini bu veriler üzerinden gerçekleştirir.

Bu kod parçacığında, x\_train ve y\_train veri kümeleri model.fit() fonksiyonuna verilir. Bu veri kümeleri, LSTM yapay sinir ağı modelinin eğitiminde kullanılacak verilerdir. model.fit() fonksiyonu ayrıca batch\_size ve epochs parametrelerini alır. batch\_size parametresi, veri kümesinin kaç parçaya bölüneceğini, epochs parametresi ise eğitim işleminin kaç kez tekrar edileceğini belirtir.

Bu kod parçacığında, batch\_size parametresine 1 ve epochs parametresine 1 değeri verilir. Bu parametreler, veri kümesinin tek parçaya bölüneceğini ve eğitim işleminin tek kez tekrar edileceğini belirtir. Bu işlem sonrasında, LSTM yapay sinir ağı modeli eğitilir ve tahmin yapmaya hazır hale gelir.

#Create the testing data set

#Create a new array containing scaled values from index 4502 to 2003

test\_data = scaled\_data[training\_data\_len - 60: , :]

#Create the data sets x test and y\_test

x\_test = []

y\_test = dataset[training\_data\_len:, :]

for i in range(60, len(test\_data)):

x\_test.append(test\_data[i-60:i, 0])

Bu kod parçacığında, modelin tahmin yapma performansını test etmek için veri kümeleri oluşturulur. Veri kümelerinin oluşturulma işlemi, eğitim verilerine benzer şekilde gerçekleştirilir.

İlk olarak, test\_data değişkenine eğitim verilerinden sonra gelen veriler atanır. Daha sonra, x\_test ve y\_test veri kümeleri oluşturulur. x\_test veri kümesi, test\_data veri kümesinden alınarak oluşturulur ve y\_test veri kümesi ise, test verilerinin hedef değişkenlerini içeren orjinal veri kümesinden alınır.

Bu kod parçacığında, x\_test veri kümesi oluşturulurken for döngüsü kullanılır. Bu döngü, test\_data veri kümesinden veri örnekleri alınarak x\_test veri kümesine eklenir. Bu işlem sonrasında, x\_test veri kümesi oluşturulmuş olur ve modelin tahmin yapma performansını test etmek için kullanılacaktır.

#convert the data to a numpy array

x\_test = np.array(x\_test)

Bu kod parçacığında, x\_test veri kümesi numpy dizisine dönüştürülür. Bu işlem, veri kümesinin tensorflow veya Keras gibi yapay sinir ağı kütüphaneleri tarafından kullanılabilir hale getirilmesi amacıyla gerçekleştirilir.

Bu kod parçacığında, x\_test veri kümesi np.array() fonksiyonu ile numpy dizisine dönüştürülür. Bu işlem sonrasında, x\_test veri kümesi numpy dizisi olarak kullanılabilir hale gelir.

#reshape the data

x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))

Bu kod parçacığında, x\_test veri kümesi reshape edilir. Bu işlem, veri kümesinin LSTM yapay sinir ağı modelinin beklediği şekle getirilmesi amacıyla gerçekleştirilir.

LSTM yapay sinir ağı modeli, veri kümesini (batch\_size, timesteps, features) şeklinde bekler. Bu nedenle, veri kümesi reshape edilerek bu şekle getirilir.

Bu kod parçacığında, x\_test veri kümesi np.reshape() fonksiyonu ile reshape edilir. Bu fonksiyon, veri kümesinin yeni şeklini belirten parametreler alır. Bu kod parçacığında, x\_test veri kümesinin (batch\_size, timesteps, features) şekline getirilmesi amacıyla gerekli parametreler verilir. Bu işlem sonrasında, x\_test veri kümesi LSTM yapay sinir ağı modelinin beklediği şekle getirilmiş olur.

#get the models predicted price values

predictions = model.predict(x\_test)

predictions = scaler.inverse\_transform(predictions)

Bu kod parçacığında, LSTM yapay sinir ağı modelinin tahminleri alınır. Bu işlem, model.predict() fonksiyonu ile gerçekleştirilir. model.predict() fonksiyonu, verilen veri kümesine göre yapay sinir ağı modelinin tahminlerini döndürür.

Bu kod parçacığında, model.predict() fonksiyonuna x\_test veri kümesi verilir. Bu işlem sonrasında, modelin x\_test veri kümesine göre yaptığı tahminler predictions değişkenine atanır.

Daha sonra, predictions değişkeni scaler.inverse\_transform() fonksiyonu ile ölçeklendirme işleminden geri döndürülür. Bu fonksiyon, verilen veri kümesini ölçeklendirme işleminden önceki haline döndürür. Bu işlem sonrasında, predictions değişkeni, LSTM yapay sinir ağı modelinin gerçek verilere göre yaptığı tahminleri içerir.

#get the root mean squared error (RMSE)

rmse = np.sqrt(np.mean(predictions - y\_test )\*\*2)

rmse

Bu kod parçacığında, LSTM yapay sinir ağı modelinin tahminlerinin doğruluk değeri hesaplanır. Bu değer, root mean squared error (RMSE) olarak adlandırılır ve modelin tahminlerinin ortalama kare hata değeri olarak hesaplanır.

RMSE değerinin hesaplanması için öncelikle, tahminler ile gerçek değerler arasındaki farkların kareleri alınır. Daha sonra, bu karelerin ortalaması alınır ve son olarak, bu ortalamanın karekökü alınır. Bu işlemler sonucunda, RMSE değeri hesaplanmış olur.

Bu kod parçacığında, np.sqrt() fonksiyonu ile karekök işlemi gerçekleştirilir ve np.mean() fonksiyonu ile karelerin ortalaması alınır. Bu işlem sonrasında, RMSE değeri elde edilir ve ekrana yazdırılır.

#Plot the data

train = data[:training\_data\_len]

valid = data[training\_data\_len: ]

valid['Predictions'] = predictions

#Visualize the data

plt.figure(figsize=(16,8))

plt.title('Model')

plt.xlabel( 'Date', fontsize=18)

plt.ylabel( 'Close Price USD ($)', fontsize=18)

plt.plot(train[ 'Close' ])

plt.plot(valid[[ 'Close', 'Predictions']])

plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')

plt.show()

Bu kod parçacığında, LSTM yapay sinir ağı modelinin tahminleri grafik olarak gösterilir. Bu grafik, modelin tahminlerinin gerçek verilere ne kadar benzediğini gösterir ve modelin doğruluk değerini gösterir.

Bu kod parçacığında, öncelikle train ve valid değişkenleri oluşturulur. Train değişkeni, modelin eğitim verisi olarak kullanılacak veri kümesini, valid değişkeni ise modelin test verisi olarak kullanılacak veri kümesini içerir. Daha sonra, valid değişkenine predictions değişkeni eklenir ve bu değişken, modelin tahminlerini içerir.

Son olarak, plt.plot() fonksiyonu ile train ve valid değişkenlerinin Close sütunları çizdirilir ve predictions değişkeni de grafikte gösterilir. Bu işlem sonrasında, modelin tahminleri gerçek verilere ne kadar benzediği görsel olarak gösterilir.

#Show the and predicted prices

Valid

Bu kod parçacığında, modelin tahminleri ile gerçek değerler gösterilir. Bu kod parçacığında, valid değişkeni yazdırılır ve bu değişken, modelin tahminleri ile gerçek değerleri içerir. Bu değişkende, Close sütunu gerçek değerleri, Predictions sütunu ise modelin tahminlerini içerir. Bu şekilde, modelin tahminlerinin gerçek değerlerle karşılaştırılması yapılabilir.