مقدمه

در نیمهٔ دوم قرن بیستم با گسترش تجارت و رشد سرمایه‌گذاری، بازار دارایی، شاهد ظهور فرضیه بازار کارآ و تئوری قیمت‌گذاری دارایی بوده است که به عنوان انقلابی در مدیریت مالی محسوب می‌شود. بازارها برای جمع‌آوری داده‌ها و انجام فرآیند اطلاعات، عقاید، تقاضاها و انتظارات، خود را با بکارگیری علوم و تکنولوژی تجهیز نموده و اجازه داده تا قیمت داراییها از طریق فرآیند اطلاعات قابل دسترس برای همه باشد. در واقع هسته قیمت‌گذاری دارایی بر اساس فرضیه بازار کارآ -که عنوان میکند به دلیل اینکه قیمت‌گذاری دارایی‌ها بر اساس تأثیرگذاری اطلاعات در دسترس همه صورت میگیرد و هیچ فردی نمیتواند سود غیرمتعارف از بازار تحصیل نماید- شکل گرفته است

اگرچه اجماع بر روی فرضیه بازار کارآ وجود ندارد، اما بسیاری از پژوهش‌های بازار نشان میدهد که حداقل اعتقاد به شکل ضعیف آن وجود دارد. از طرفی بعضی از پژوهشگران ادعای بینظمی در بازار را دارند و معتقدند که بازار بر اساس یک فرآیند غیرخطی که بصورت تصادفی نمایان میشود، شکل میگیرد. بنابراین دلالت بر تصادفی بودن بازار و آشوب حاکم بر آن دارن .

فرآیند قطعی میتواند از طریق ویژگی‌های آن و با استفاده از ابزارهایی مانند رگرسیون مدل پیش‌بینی شود. فرآیند تصادفی نیز ممکن است با استفاده از پارامترهای آماری از یک عملکرد توزیع (تابع عضویت) قابل مدل نمودن وپیش‌بینی باشد. بنابراین با استفاده از فقط یک تکنیک قطعی یا آماری نمیتوان جواب‌گوی سیستم پیچیده، غیردقیق و با دانش مبهم بود .ابزارهای جدید از جمله هوش مصنوعی به علت قابلیت استفاده از تکنیکهای قطعی و آماری و همچنین به علت قابلیت یادگیری نقشه‌های غیرخطی بین داده‌ها توانایی گرفتن هر دو ویژگی قطعی و تصادفی را دارا هستند و امکان مدل نمودن و پیش‌بینی قیمت بازار سهام تاحدی میسر مینماید.

یکی از شاخه‌های وسیع و پرکاربرد هوش مصنوعی، یادگیری ماشین است که به مطالعه‌ی شیوه و الگوریتم‌هایی میپردازد که براساس آن رایانه‌ها و سامانه‌ها توانایی یادگیری انجام اعمال مختلف را پیدا میکنند. در واقع یادگیری حاصل بر اساس نوعی مشاهده یا داده است و شامل روش‌هایی است که برای دسته‌بندی داده‌ها استفاده می‌شوند.

ساختار پروژه به شرح زیر است :

طرح موضوع : در این بخش در رابطه با هدف پژوهش صحبت می‌کنیم .

روش شناسی پژوهش : در این بخش روشهای اقتصادی و هوشمند یادگیری ماشین مورد استفاده را به تفصیل توضیح میدهد.

تجزیه و تحلیل داده‌ها : در این بخص با استفاده از پایتون به تجزیه و تحلیل داده‌ها یعنی مصورسازی و اجرای الگوریتم‌های یادگیری ماشین میپردازیم.

نتیجه‌گیری : در این بخش نتایج بهدست آمده مورد تحلیل و بررسی قرارگرفته و پیشنهادهایی برای ادامه‌ی کار در آینده ارائه شده است.

طرح موضوع

وجود الگو در تغییر قیمت‌ها در بازار سرمایه وکشف روابط بین قیمت و سایر متغیرها با استفاده از مدل‌های مناسب، یکی از دغدغه‌های بازار سرمایه است چون اگر بتوان الگوی تغییر قیمت را در بازار کشف کنیم میتوانیم به کمک آن الگو، استراتژی معاملاتی طراحی کنیم و به سود خوبی برسم.

پس دغدغه اصلی همه افراد حاضر در بازار سرمایه پیش بینی قیمت سهام است، روش‌های مختلفی برای پیش‌بینی قیمت سهام وجود دارد مانند

تحلیل تکنیکال : تحلیلی که با استفاده از قیمت و روند سهام در گذشته به پیش‌بینی آینده میپردازد.

تحلیل بنیادی : تحلیلی که با مطالعه‌ی اطلاعات ذاتی سهام به‌طور مثال، اطلاعات مالی سهم مربوطه به پیش‌بینی آینده میپردازد.

ما در این پژوهش سعی داریم که به کمک روش‌های مختلف یادگیری ماشین به پیش‌بینی قیمت(جهت قیمت) جفت ارز **EURUSD** بپردازیم.

فرضیه پژوهش

جهت قیمت جفت ارز **EURUSD** با توجه به نوع بازار و متغیرهای زیاد قابل مشاهده تا حدودی قابل پیش بینی است.

روش شناسی پژوهش

پژوهش حاضر به منظور دستیابی به یک الگو و مدل برای پیشبینی قیمت جفت ارز انجام میپذیرد، از حیث اینکه نوعی ابزار و نوعی الگو برای مدیران و سرمایه‌گذاران است و میتواند در تصمیم گیری مورد استفاده قرار گیرد؛ این تحقیق کاربردی محسوب میشود، و از طرفی به لحاظ اینکه یکی از مشکلات و مسائل پیش روی را حل میکند، میتواند پژوهشی علمی نیز باشد.

فرآیند تحقیق

۱. روش جمع‌آوری داده‌ها و تعریف متغیرها

داده‌هایی که در این پژوهش مورد استفاده قرار گرفته مربوط به جفت ارز **EURUSD** میباشد که در بازار **Forex** معامله میشود.

داده‌های ما شامل اطلاعات قیمتی جفت ارز **EURUSD** در تایم فریم ۱ روزه و بازه زمانی ۶ ساله یعنی از سال (۲۰۱۶_۲۰۲۱) می‌باشد که با استفاده از زبان برنامه نویسی **MQL4** در محیط برنامه متاترید ۴ استخراج شده است.

در زیر لیست متغیرهای ما قابل مشاهده است:

نام متغیر	شرح
Date	تاریخ
Lag1	بازدهی روز قبل $(Close_{Yesterday} - Open_{Yesterday})$
Lag2	بازدهی ۲ روز قبل(میزان تغییر قیمت در ۲ روز پیش)
Lag3	درصد بازده 3 روز قبل

Lag4	درصد بازده 4 روز قبل
Lag5	درصد بازده 5 روز قبل
RSI	اندیکاتور شاخص قدرت نسبی
StdDev	اندیکاتور مقیاس تغییرات قیمت مربوط به میانگین متحرک (Standard Deviation)
Momentum	اندیکاتور شناسایی سرعت حرکت قیمت
Macd	اندیکاتور همگرایی (Convergence) و واگرایی (Divergence) میانگین متحرک
Atr	اندیکاتور نوسان گیر (Average True Range)
Sar	اندیکاتور پیروی روند Parabolic SAR
candelstatus	وضعیت قیمت سهام (اگر قیمت مثبت باشد مقدار ۱ و اگر قیمت منفی باشد مقدار ۰)

۲. روش‌های یادگیری ماشین مورد استفاده

k-nearest neighbors algorithm

یک متد آمار ناپارامتری است که برای طبقه‌بندی آماری و رگرسیون استفاده می شود. در هر دو حالت K شامل نزدیک ترین مثال آموزشی در فضای داده ای می باشد و خروجی آن بسته به نوع مورد استفاده در طبقه بندی و رگرسیون متغیر است. در حالت طبقه بندی با توجه به مقدار مشخص شده برای K، به محاسبه فاصله نقطه ای که میخواهیم برچسب آن را مشخص کنیم با نزدیک ترین نقاط میپردازد و با توجه به تعداد رای حداکثری این نقاط همسایه، در رابطه با برچسب نقطه مورد نظر تصمیم گیری می‌کنیم. برای محاسبه این فاصله میتوان از روش های مختلفی استفاده کرد که یکی از مطرح ترین این روش ها، فاصله اقلیدسی است. در حالت رگرسیون نیز میانگین مقادیر بدست آمده از K خروجی آن می باشد. از آنجا که محاسبات این الگوریتم بر اساس فاصله است نرمال‌سازی داده‌ها می‌تواند به بهبود عملکرد آن کمک کند.

روش نزدیکترین K همسایه، یک گروه شامل K داده از مجموعه‌ی داده‌های آموزشی که نزدیکترین داده‌ها به داده‌ی ورودی باشند را انتخاب کرده و براساس برتری دسته یا برچسب مربوط به آنها در مورد دسته‌ی داده‌ی آزمایشی مزبور تصمیم‌گیری می‌نماید. به عبارت ساده‌تر این روش دسته‌ای را انتخاب میکند که در همسایگی انتخاب‌شده بیشترین تعداد داده، منتسب به آن دسته باشند، بنابراین دسته‌ای که از همه‌ی رده‌ها بیشتر، در بین K نزدیکترین همسایه مشاهده شود، به‌عنوان دسته‌ی داده‌ی جدید در نظر گرفته میشود

مراحل الگوریتم knn شامل موارد زیر است :

۱. داده‌ها را به دو بخش Test , train تقسیم می‌کنیم
۲. فاصله نقطه(نقاط) درون test را از تمام مشاهده درونtrain محاسبه میکنیم.
۳. فاصله‌های بدست آمده را بصورت صعودی مرتب میکنم

۴. K نقطه نزدیک به test را در نظر میگیریم.(k همسایه)
۵. بسته به نوع داده‌ها خروجی را بر اساس K همسایه مشخص میکنیم. اگر متغیر پاسخ ما از نوع رسته‌ای باشند با استفاده از قاعده تصمیم اکثریت و اگر متغیر پاسخ ما کیفی باشد با استفاده از متوسط‌گیری داده‌های محلی در رابطه با خروجی نظر می‌دهیم.

انتخاب K :

بهترین انتخاب K بستگی به داده‌ها دارد؛ به‌طور کلی، مقادیر بزرگ K باعث کاهش خطا در طبقه‌بندی می‌شود، اما وضوح مرز بین کلاس‌ها را کمتر می‌کند.

K مناسب را می‌توان با استفاده از تکنیک‌های مختلف انتخاب کرد. اگر K=1 در این صورت، الگوریتم نزدیکترین همسایه نامیده می‌شود.

Decision Tree

یکی از پرکاربردترین الگوریتم‌های داده‌کاوی، الگوریتم درخت تصمیم است. در داده‌کاوی، درخت تصمیم یک مدل پیش‌بینی کننده است به طوری که می‌تواند برای هر دو مدل رگرسیون و طبقه‌ای مورد استفاده قرار گیرد. زمانی که درخت برای کارهای طبقه‌بندی استفاده می‌شود، به عنوان درخت طبقه‌بندی (Classification Tree) شناخته می‌شود .

الگوریتم CART پایه ای برای الگوریتم های مهمی مانند درختان تصمیم کیسه ای، جنگل تصادفی و درختان تصمیم تقویت شده فراهم می کند.

نکته حائز اهمیت این است که این الگوریتم درخت‌های باینری ایجاد می‌کند به طوری که از هر گره داخلی دو لبه از آن خارج می‌شود و درخت‌های بدست آمده توسط روش اثربخشی هزینه، هرس می‌شوند.

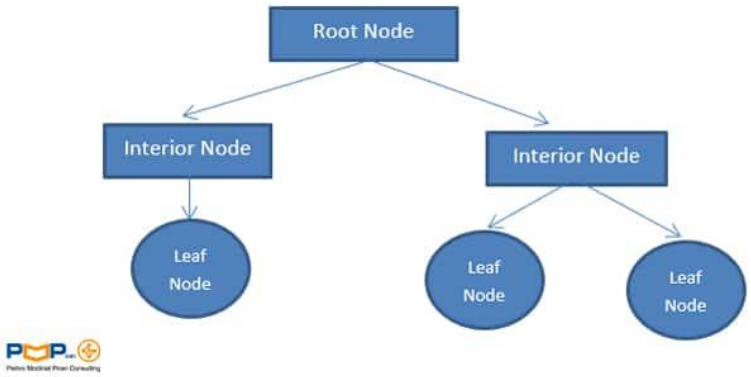
اجزای اصلی درخت تصمیم
برگ (Leaf Nodes): گره‌هایی که تقسیم‌های متوالی در آنجا پایان می‌یابد. برگ‌ها با یک کلاس مشخص می‌شوند.

ریشه (Root Node): منظور از ریشه، گره آغازین درخت است.

شاخه (Branches) :در هر گره داخلی به تعداد جواب‌های ممکن شاخه ایجاد می‌شود.

هر گره در درخت تصمیم نشان‌دهنده‌ی یک ویژگی است و هر شاخه از هر گره نشان‌دهنده‌ی مقادیر ممکن برای ویژگی مرتبط با آن گره است، هر نمونه با شروع از ریشه دسته‌بندی میشود، مقدار ویژگی موجود در ریشه برای نمونه‌ی مورد نظر چک میشود و با

توجه به مقدار ویژگی ریشه برای نمونه، داده‌ی ورودی به زیردرخت مناسب درخت پیشروی میکند .این فرآیند برای این زیردرخت نیز تکرار میشود (شکل ۱) نمایی از یک درخت تصمیم را نشان میدهد.

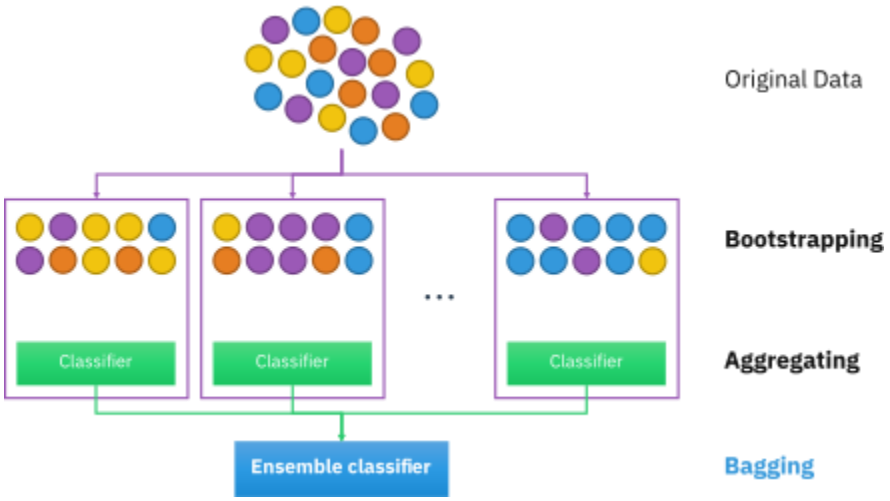


Bootstrap Aggregation

یکی از ایرادات درخت رگرسیونی واریانس بالای آن است، یکی از روش‌هایی که به ما کمک میکند درخت را بهبود ببخشیم **Bagging** است.

Bagging (Bootstrap Aggregation) زمانی استفاده می شود که هدف ما کاهش واریانس درخت تصمیم باشد.

در اینجا ایده این است که چندین زیرمجموعه از داده ها را از نمونه آموزشی که به طور تصادفی با جایگزینی انتخاب شده اند ایجاد کنیم. اکنون، هر مجموعه ای از داده های زیرمجموعه برای آموزش درخت تصمیم خود استفاده می شود. در نتیجه، ما با مجموعه ای از مدل های مختلف مواجه می شویم. میانگین تمام پیش بینی ها از درختان مختلف استفاده می شود که از یک درخت تصمیم گیری قوی تر است.



Random Forest

جنگل تصادفی یا جنگل‌های تصمیم تصادفی یک روش یادگیری ترکیبی برای دسته‌بندی، رگرسیون می‌باشد، که بر اساس ساختاری متشکل از شمار بسیاری درخت تصمیم، بر روی زمان آموزش و خروجی کلاس‌ها (کلاس‌بندی) یا برای پیش‌بینی‌های هر درخت به شکل مجزا، کار می‌کنند. جنگل‌های تصادفی برای درختان تصمیم که در مجموعه آموزشی دچار بیش برازش می‌شوند، مناسب هستند. عملکرد جنگل تصادفی معمولا بهتر از درخت تصمیم است، اما این بهبود عملکرد تا حدی به نوع داده هم بستگی دار

Random Forest یک مرحله بیشتر از **Bagging** دارد.

این یک مرحله اضافی علاوه بر گرفتن زیرمجموعه تصادفی داده ها، به جای استفاده از همه ویژگی ها برای رشد درختان، به انتخاب تصادفی ویژگی ها نیز نیاز است.

الگوریتم جنگل تصادفی :

۱. فرض کنید **N** مشاهدات و **M** ویژگی در مجموعه داده های آموزشی وجود دارد. ابتدا نمونه ای از مجموعه داده های آموزشی به صورت تصادفی با جایگزینی گرفته می شود.

۲. زیرمجموعه‌ای از ویژگی‌های **M** به‌طور تصادفی انتخاب می‌شوند و هر ویژگی که بهترین تقسیم را ارائه می‌دهد برای تقسیم مجدد گره استفاده می‌شود.

۳. درخت به بزرگترین رشد می کند.

۴. مراحل بالا تکرار شده و پیش بینی بر اساس تجمیع پیش بینی ها از **n** تعداد درخت داده می شود.

Linear Discriminant Analysis

روش‌های آماری هستند که از جمله در یادگیری ماشین و بازشناخت الگو برای پیدا کردن ترکیب خطی خصوصیاتی که به بهترین صورت دو یا چند کلاس از اشیا را از هم جدا می‌کند، استفاده می‌شوند.

آنالیز تشخیصی خطی بسیار به تحلیل واریانس و تحلیل رگرسیونی نزدیک است؛ در هر سه این روش‌های آماری متغیر وابسته به صورت یک ترکیب خطی از متغیرهای دیگر مدل‌سازی می‌شود .با این حال دو روش آخر متغیر وابسته را از نوع فاصله‌ای در نظر می‌گیرند در حالی که آنالیز افتراقی خطی برای متغیرهای وابسته اسمی یا رتبه‌ای به کار می‌رود. از این رو آنالیز افتراقی خطی به رگرسیون لجستیک شباهت بیشتری دارد.

آنالیز تشخیصی خطی همچنین با تحلیل مؤلفه‌های اصلی و تحلیل عاملی هم شباهت دارد؛ هر دوی این روش‌های آماری برای ترکیب خطی متغیرها به شکلی که داده را به بهترین نحو توضیح بدهد به کار می‌روند. یک کاربرد عمده هر دوی این روش‌ها،

کاستن تعداد بعدهای داده است. با این حال این روش‌ها تفاوت عمده‌ای با هم دارند: در آنالیز افتراقی خطی، تفاوت کلاس‌ها مدل‌سازی می‌شود در حالی که در تحلیل مؤلفه‌های اصلی تفاوت کلاس‌ها نادیده گرفته می‌شود.

LDAارتباط نزدیکی با تحلیل واریانس و تحلیل رگرسیون دارد که سعی دارند یک متغیر مستقل را به عنوان ترکیبی خطی از ویژگی‌های دیگر بیان کنند. این متغیر مستقل در LDA به شکل برچسب یک کلاس است. همچنین LDA ارتباطی تناتنگ با تحلیل مؤلفه‌های اصلی PCA دارد. چرا که هر دو متد به دنبال ترکیبی خطی از متغیرهایی هستند که به بهترین نحو داده‌ها را توصیف می‌کنند LDA .همچنین سعی در مدلسازی تفاوت بین کلاس‌های مختلف داده‌ها دارد. از LDA زمانی استفاده می‌شود که اندازه‌های مشاهدات، مقادیر پیوسته باشند.

QDA واقعاً تفاوت چندانی با LDA ندارد به جز اینکه شما فرض می کنید که ماتریس کوواریانس می تواند برای هر کلاس متفاوت باشد و بنابراین ماتریس کوواریانس را به طور جداگانه برای هر کلاس تخمین می زنیم.

SVM

یکی از روش‌های یادگیری بانظارت است که از آن برای طبقه‌بندی و رگرسیون استفاده می‌کنند.

این روش از جمله روش‌های نسبتاً جدیدی است که در سال‌های اخیر کارایی خوبی نسبت به روش‌های قدیمی‌تر برای طبقه‌بندی نشان داده‌است. مبنای کاری دسته‌بندی کننده SVM دسته‌بندی خطی داده‌ها است و در تقسیم خطی داده‌ها سعی می‌کنیم خطی را انتخاب کنیم که حاشیه اطمینان بیشتری داشته باشد. حل معادله پیدا کردن خط بهینه برای داده‌ها به وسیله روش‌های QPکه روش‌های شناخته شده‌ای در حل مسائل محدودیت‌دار هستند صورت می‌گیرد. قبل از تقسیمِ خطی برای اینکه ماشین بتواند داده‌های با پیچیدگی بالا را دسته‌بندی کند داده‌ها را به وسیله تابعِ phi به فضای با ابعاد خیلی بالاتر می‌بریم. برای اینکه بتوانیم مسئله ابعاد خیلی بالا را با استفاده از این روش‌ها حل کنیم از قضیه دوگانی لاگرانژ[برای تبدیلِ مسئله مینیمم‌سازی مورد نظر به فرم دوگانی آن که در آن به جای تابع پیچیده phi که ما را به فضایی با ابعاد بالا می‌برد، تابعِ ساده‌تری به نامِ تابع هسته که ضرب برداری تابع phi است ظاهر می‌شود استفاده می‌کنیم. از توابع هسته مختلفی از جمله هسته‌های نمایی، چندجمله‌ای و سیگموید می‌توان استفاده نمود.

اگر از هسته با تابع گوسین استفاده شود، فضای ویژگی متناظر، یک فضای هیلبرت نامتناهی است. دسته‌کننده بیشترین حاشیه، خوش ترتیب است، بنابراین ابعاد نامتناهی، نتیجه را خراب نمی‌کند.

تجزیه و تحلیل داده‌ها

در این بخش ما به کمک پایتون به تجزیه و تحلیل داده‌ها میپردازیم

Load datasets

```
EURUSD_df = pd.read_csv("Datasets/Data_EURUSD.csv")

EURUSD_df.shape
```

(1553, 13)

داده‌های خود را فراخوانی کردیم.
ابعاد داده‌های ما ۱۵۵۳ *۱۳ است، یعنی برای ۱۵۵۳ مشاهده(هر مشاهده یک روز معاملاتی است) ۱۳ متغیر که در قبل توضیح دادیم اندازه گیری شده است.

در پایین برای ۴ مشاهده ابتدایی مقادیر متغیرها را مشاهده میکنید.

```
EURUSD_df.head()
```

	Date	lag1	lag2	lag3	lag4	lag5	RSI	StdDev	Momentum	macd	atr	sar	candelstatus
0	2016.01.08	0.01491	0.00385	-0.00883	-0.00425	-0.00663	53.980341	0.007385	100.729961	0.001760	0.009561	1.097535	0
1	2016.01.11	-0.00032	0.01491	0.00385	-0.00883	-0.00425	53.304457	0.007288	100.053144	0.001487	0.009937	1.070710	0
2	2016.01.12	-0.00608	-0.00032	0.01491	0.00385	-0.00883	49.097754	0.006899	99.107363	0.001160	0.009863	1.071234	0
3	2016.01.13	-0.00018	-0.00608	-0.00032	0.01491	0.00385	48.866270	0.006218	99.502264	0.000835	0.009872	1.071748	1
4	2016.01.14	0.00215	-0.00018	-0.00608	-0.00032	0.01491	50.456963	0.006051	99.306101	0.000523	0.010026	1.072251	0

در زیر لیست متغیرهای ما قابل مشاهده است.

```
EURUSD_df.columns
```

```
Index(['Date', 'lag1', 'lag2', 'lag3', 'lag4', 'lag5', 'RSI', 'StdDev',
       'Momentum', 'macd', 'atr', 'sar', 'candelstatus'],
      dtype='object')
```

متغیر تاریخ (Date) در محاسبات ما نقشی ندارد پس آن را از مجموعه داده خود حذف میکنیم :

```
EURUSD_df = EURUSD_df.drop(columns=['Date'])

EURUSD_df.columns
```

```
Index(['lag1', 'lag2', 'lag3', 'lag4', 'lag5', 'RSI', 'StdDev', 'Momentum',
       'macd', 'atr', 'sar', 'candelstatus'],
      dtype='object')
```

برسی مقادیر گمشده

```
EURUSD_df.shape[0] - EURUSD_df.dropna().shape[0]
```

```
0
```

ما در مجموعه داده خود هیچ مقدار گمشده‌ای نداریم پس نیاز نیست در رابطه با مقادیر گمشده عملیات خاصی اجرا کنیم.

نرمال سازی داده ها

```
list_df = []

for df in [EURUSD_df]:
    columns = df.columns

    y = df["candelstatus"].copy()
    X = df.drop(columns=["candelstatus"]).copy()
    scaler = StandardScaler()
    X = pd.DataFrame(scaler.fit_transform(X))
    X["candelstatus"] = np.array(y)
    X.columns = columns
    list_df.append(X)

EURUSD_df = list_df[0]
```

یکی از عملیات‌های مهم نرمال سازی داده‌ها است، چون متغیرهای ما دارای واحدهای اندازه‌گیری مختلفی هستند پس واجب است که انها را هم مقیاس کنیم که ما با استفاده از کدهای فوق به هدف خود یعنی هم مقیاس کردن متغیرها میرسیم.

corralation matrix

```
corr_EURUSD = EURUSD_df.drop(columns=["candelstatus"]).corr()
corr_EURUSD
```

	lag1	lag2	lag3	lag4	lag5	RSI	StdDev	Momentum	macd	atr	sar
lag1	1.000000	-0.007744	0.011555	-0.011621	-0.012631	0.350008	-0.005304	0.258972	-0.007602	-0.015390	-0.088667
lag2	-0.007744	1.000000	-0.008421	0.011000	-0.012228	0.317634	0.003791	0.256290	0.002976	-0.012359	-0.089120
lag3	0.011555	-0.008421	1.000000	-0.007187	0.012476	0.298527	0.013321	0.253319	0.019192	-0.011459	-0.074985
lag4	-0.011621	0.011000	-0.007187	1.000000	-0.006348	0.273546	0.014181	0.261249	0.039874	-0.003227	-0.061980
lag5	-0.012631	-0.012228	0.012476	-0.006348	1.000000	0.250988	0.016851	0.269084	0.064127	0.001833	-0.032040
RSI	0.350008	0.317634	0.298527	0.273546	0.250988	1.000000	0.089176	0.845943	0.638840	0.130709	0.015656
dDev	-0.005304	0.003791	0.013321	0.014181	0.016851	0.089176	1.000000	0.058699	0.086920	0.608300	-0.053347
ıtum	0.258972	0.256290	0.253319	0.261249	0.269084	0.845943	0.058699	1.000000	0.420508	0.037559	-0.047431
nacd	-0.007602	0.002976	0.019192	0.039874	0.064127	0.638840	0.086920	0.420508	1.000000	0.172637	0.248710
atr	-0.015390	-0.012359	-0.011459	-0.003227	0.001833	0.130709	0.608300	0.037559	0.172637	1.000000	-0.039402
sar	-0.088667	-0.089120	-0.074985	-0.061980	-0.032040	0.015656	-0.053347	-0.047431	0.248710	-0.039402	1.000000

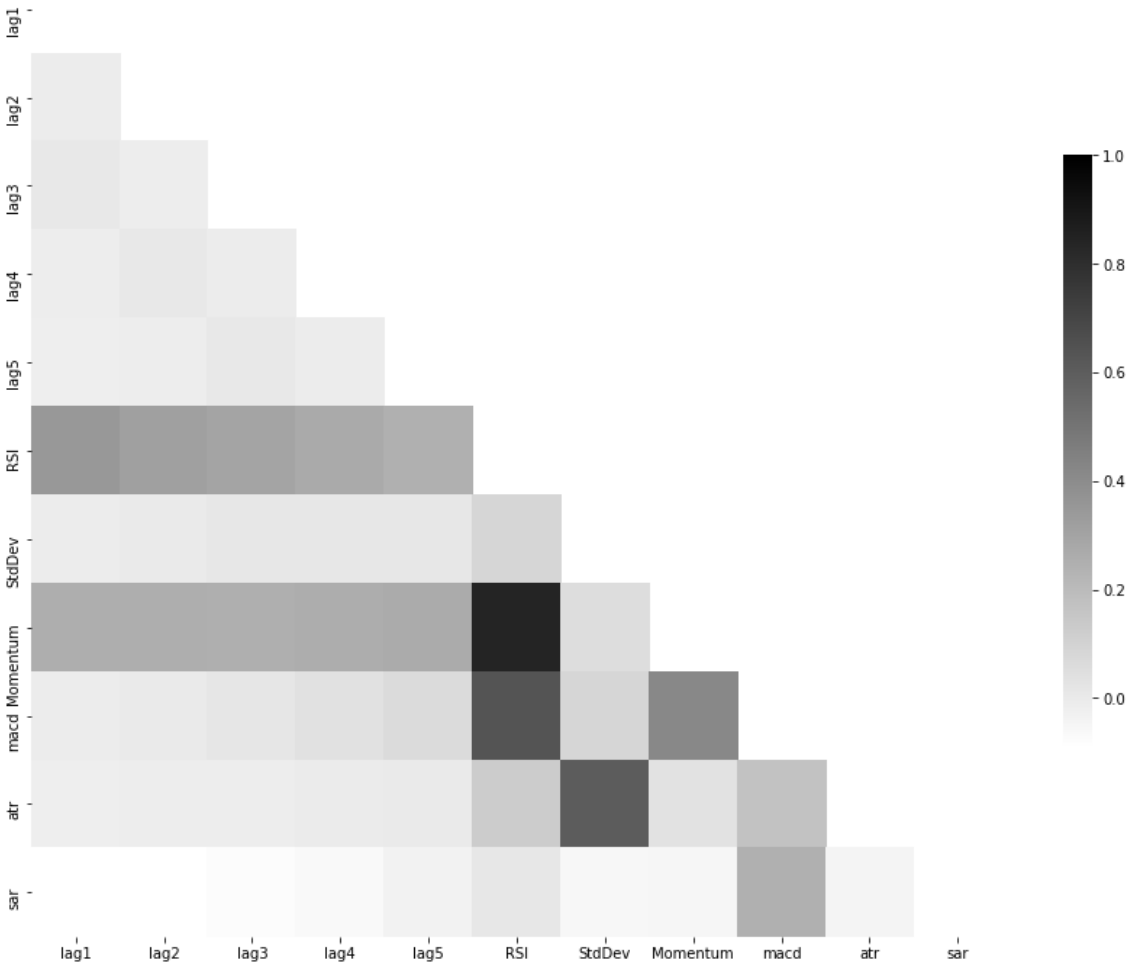
با استفاده از کد فوق ماتریس همبستگی بین متغیرها را بدست می آوریم.

قطر اصلی این ماتریس ۱ است چون میزان همبستگی هر متغیر با خودش ۱۰۰ درصد است.

برای این که بهتر بتوانیم همبستگی و وابستگی بین متغیرها را درک کنیم و راحت تر ان را مشاهده کنیم میتوانیم از **heatmap** استفاده کنیم :

heatmap

```
fig = plt.figure(figsize=(15, 25))
ax = fig.add_subplot(111)
mask=np.zeros_like(corr_EURUSD)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr_EURUSD,
          xticklabels=corr_EURUSD.columns,
          yticklabels=corr_EURUSD.columns,
         ax=ax, mask=mask, square=True,
          cbar_kws={"shrink": 0.3},
         cmap="binary")
plt.show()
```



در این ماتریس هر چه رنگ مکعبی تیره تر باشد همبستگی بین آن متغیرها بیشتر است.

برای مثال همبستگی RSI و Momentume تقریبا برابر ۰.۸ است که این یعنی از همبستگی بالایی برخوردار هستند.

مشاهده میشود که همبستگی بین متغیرهای lag1 lag2 lag3 lag4 lag5 و متغیر sar بسیار پایین است و تقریبا صفر است.

در کل میتوان گفت که همبستگی بین متغیرهای ما پایین است .

Class Distribution

```
EURUSD_df["candelstatus"].value_counts()
```

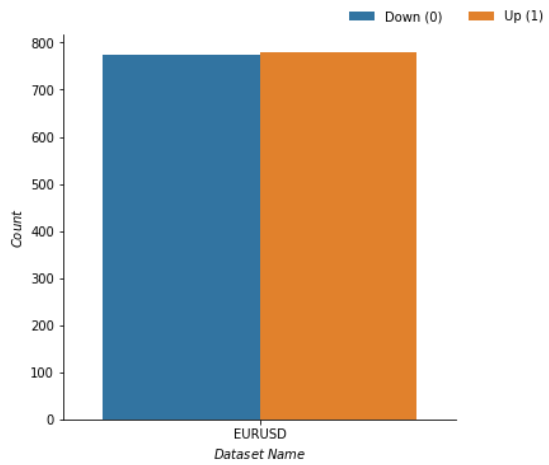
```
1   779
0   774
Name: candelstatus, dtype: int64
```

Class distribution plot

```
EURUSD_class = EURUSD_df[["candelstatus"]]
EURUSD_class["dataset_name"] = ["EURUSD" for i in range(EURUSD_df.shape[0])]
```

```
merge_df = pd.concat([EURUSD_class], ignore_index=True)
g = sns.catplot(x='dataset_name', hue='candelstatus',
          kind='count', data=merge_df, ci=False, aspect=1,
            legend=False)
plt.xlabel("$Dataset$ $Name$")
plt.ylabel("$Count$")
plt.legend(labels=["Down (0)", "Up (1)"],
        bbox_to_anchor=[0.7,1], ncol=5, frameon=False)

plt.show()
```



متغیر پاسخ ما که مثبت یا منفی بودن سهام در روز جاری است دارای دو وضعیت ۰ و ۱ است.

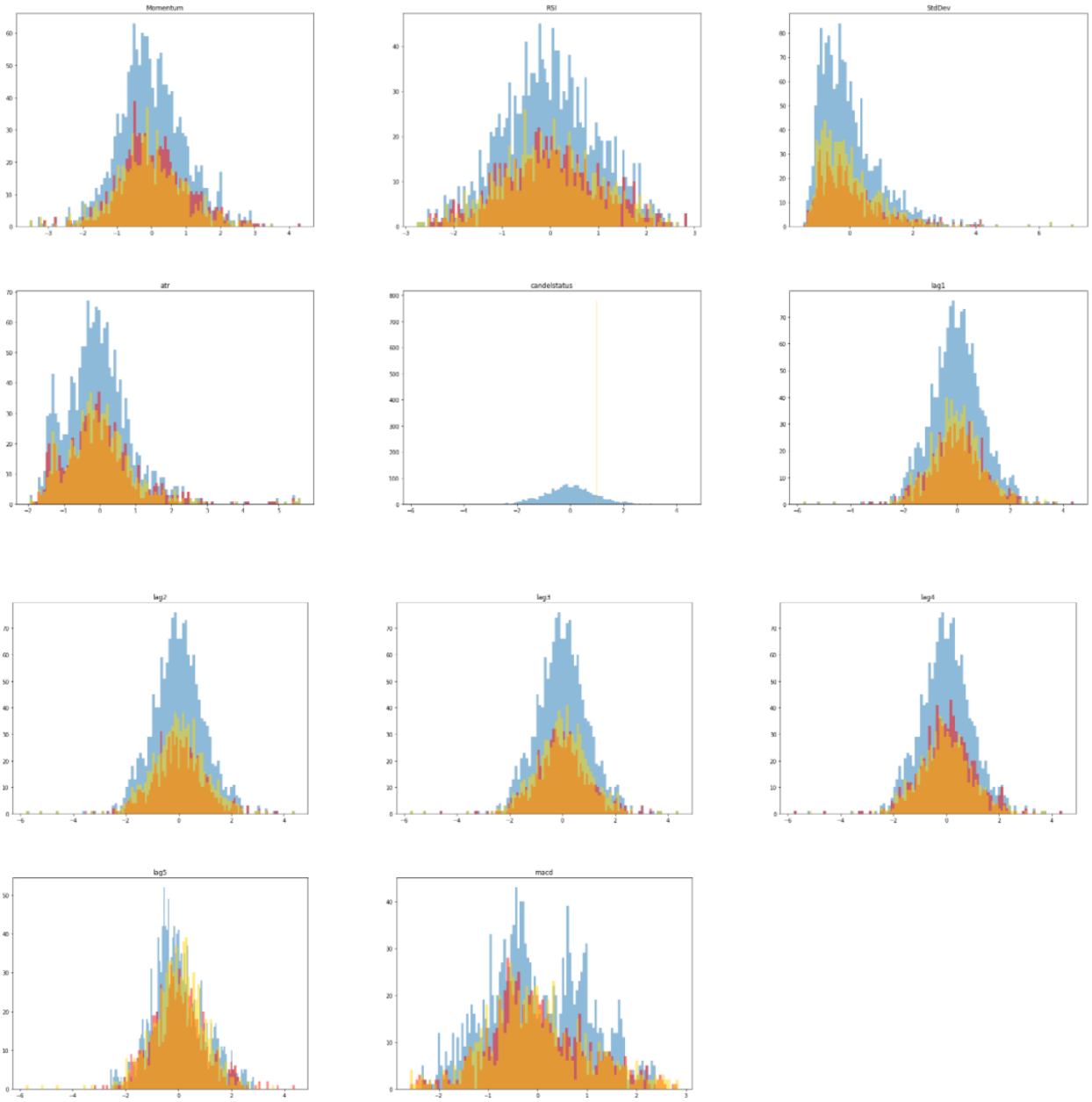
شکل ۱ توزیع های کلاس «بالا» و «پایین» را برای همه مجموعه داده ها به تصویر می کشد

ما متوجه شدیم که کلاس‌ها تقریباً متعادل هستند.

از ۱۵۵۳ روزی که اطلاعات استخراج کردیم ۷۷۹ روز وضعیت سهام مثبت است یعنی مقدار ۱ و ۷۷۴ روز دیگر قسمت سهام منفی است.

Histogram plot of features

```
ax = EURUSD_df.drop(columns=["candelstatus"]).hist(bins=100, alpha=0.5, label="Full", grid=False, figsize=(35, 35))
EURUSD_df[EURUSD_df["candelstatus"] == 0].hist(bins=100, ax=ax.ravel()[:12],grid=False, color="Red", alpha=0.5, label="0")
EURUSD_df[EURUSD_df["candelstatus"] == 1].hist(bins=100, ax=ax.ravel()[:12], color="gold",grid=False, alpha=0.5, label="1")
plt.show()
```



شکل ۴ هیستوگرام های توزیع مقادیر ویژگی را برای همه ویژگی ها نشان می دهد.

برای هر ویژگی سه هیستوگرام را نشان می دهد:

هیستوگرام آبی : تمام نمونه های موجود در مجموعه داده را در نظر می گیرد.

هیستوگرام قرمز : فقط نمونه های کلاس «Down(0)» را در نظر می گیریم.

هیستوگرام طلایی : فقط نمونه های کلاس "Up(1)" را در نظر می گیریم.

بر اساس شکل، دو مشاهده انجام شد.

بیشتر ویژگی‌ها دارای مقادیری بودند که حول میانگین توزیع شده بودند، که نشان می‌دهد تبدیل لگاریتمی برای این ویژگی‌ها غیرضروری است.

با این حال، برخی از ویژگی ها مانند StdDev,Sar مقداری چوله هستند، یعنی از دارای توزیع نرمال نیستند و تبدیل میتواند برای آنها مفید باشد. ولی در تحقیقی که قبلا انجام دادیم تبدیل لگاریتمی زیاد تغیر خاصی روی نتایج نداشته است، پس ما در اینجا نیز تبدیل را اعمال نمیکنیم. در نتیجه، ما به استفاده از مجموعه داده اصلی عملکرد را میسنجیم.

اگر هیستوگرام متغیری شامل نمونه‌هایی از کلاس‌های مختلف (قرمز و طلایی) همپوشانی داشته باشند و میانگین‌های مشابهی نیز داشته باشند این نشان میدهد که آن متغیر یا ویژگی به تشخیص بین کلاس ها کمک نمی کند.

Machine Learning Algorithms

در این بخش ما به اجرای الگوریتم‌هایی که در قبل توضیح دادیم میپردازیم

اولین کاری که ما باید انجام بدهیم این است که داده‌های خود را به دو بخش Test و Train تقسیم کنیم

Test and Trian

```
X_train, X_test, y_train, y_test = train_test_split(EURUSD_df.drop(columns=["candelstatus"]) ,EURUSD_df.candelstatus
                                   ,stratify=None,test_size=0.2, shuffle=False)
```

```
X_train.shape
```

(1242, 11)

```
X_test.shape
```

(311, 11)

۱۲۴۲ داده ابتدایی را به عنوان داده آموزش در نظر میگیریم یعنی ۸۰ درصد از داده‌ها.

۲۰ درصد پایانی داده‌ها که شامل ۳۱۱ مشاهده میشود را به عنوان داده تست در نظر میگیریم.

k-nearest neighbors algorithm

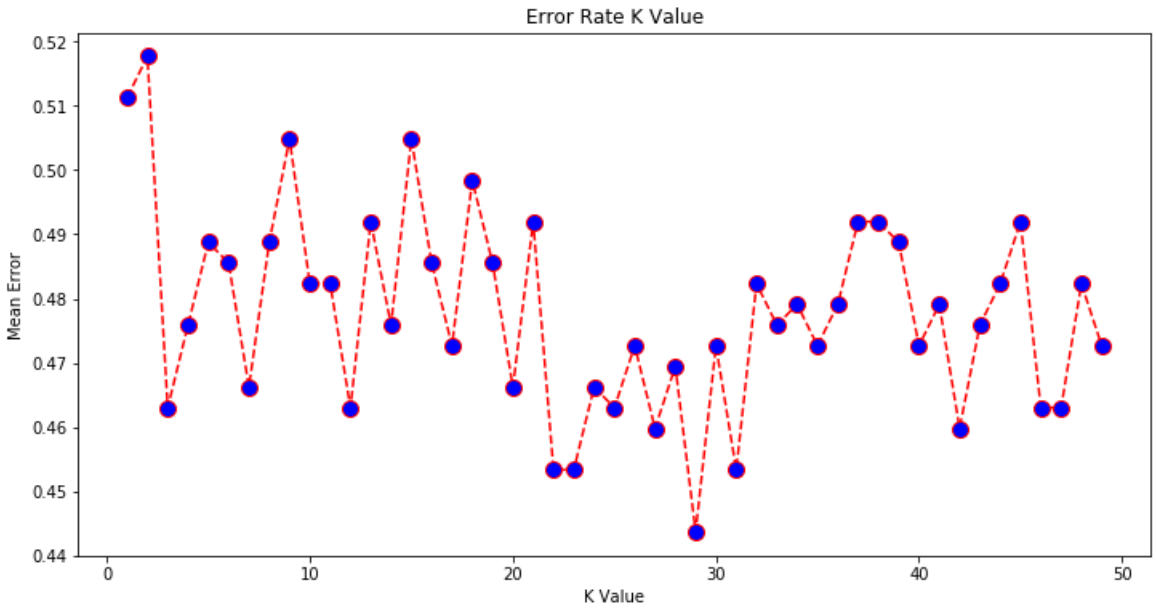
```
from sklearn.neighbors import KNeighborsClassifier
```

```
error = []

# Calculating error for K values between 1 and 50
for i in range(1, 50):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 50), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

.



مهم ترین مرحله در اجرای **KNN** پیدا کردن بهترین **K** است.

میدانیم که هیچ راهی وجود ندارد که از قبل بدانیم کدام مقدار **K** بهترین نتیجه را دارد.

یکی از راه‌هایی که به ما کمک می‌کند بهترین مقدار **K** را پیدا کنید، رسم نمودار مقدار **K** و نرخ خطای مربوطه برای مجموعه داده است.

در این بخش، میانگین خطای مقادیر پیش‌بینی‌شده مجموعه تست را برای همه مقادیر **K** بین ۱ تا ۵۰ رسم کردیم.

مشاهده میشود که کمترین خطا مربوط به **K=29** است، یعنی برای پیش بینی لیبل داده‌های تست خود بهتر است که از ۲۹ همسایه نزدیک کمک بگیریم.

یعنی مثلا میخواهیم لیبل اولین داده تست خود را پیش بینی کنیم، برای اینکار به ۲۹ مشاهده‌ای که به آن داده نزدیک است نگاه میکنیم و طبق قاعده رای اکثریت در رابطه با لیبل داده تست نظر میدهیم.

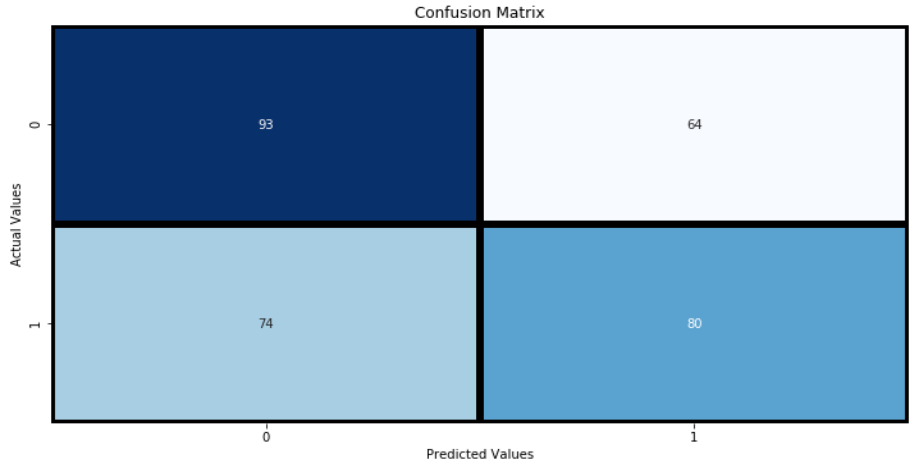
اکنون که بهترین K را انتخاب کردیم، به کمک آن پیش بینی میکنیم :

Predict

```
classifier = KNeighborsClassifier(n_neighbors=29)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
cm=confusion_matrix(y_test,y_pred)
plt.figure(figsize=(12,6))
plt.title("Confusion Matrix")
sns.heatmap(cm, annot=True,fmt='d', cmap='Blues')
plt.ylabel("Actual Values")
plt.xlabel("Predicted Values")
plt.savefig('confusion_matrix.png')
```




```
test_acc = accuracy_score(y_test, y_predKNN)
print("The Accuracy for Test Set is {}".format(test_acc*100))
```

```
The Accuracy for Test Set is 55.62700964630225
```

```
print(classification_report(y_test, y_predKNN))
```

```
              precision    recall  f1-score   support

           0       0.56      0.59      0.57       157
           1       0.56      0.52      0.54       154

    micro avg       0.56      0.56      0.56       311
    macro avg       0.56      0.56      0.56       311
 weighted avg       0.56      0.56      0.56       311
```

در جدول فوق تعداد پیش بینی‌های درست و غلط را مشاهده میکنیم.

برای مثال ما از ۱۵۷ روز منفی(۰) ۹۳ روز را درست پیش بینی کردیم و ۶۴ روز را به اشتباه مثبت(۱) پیش بینی کردیم.

۷۴ روز از روزهایی که سهام مثبت(۱) بوده را به اشتباه منفی(۰) پیش بینی کردیم.

در مجموع از ۳۱۱ روزی که تست کردیم الگوریتم KNN توانست ۱۷۴ روز را درست پیش بینی کند یعنی دقت ما ۵۵.۶٪ است.

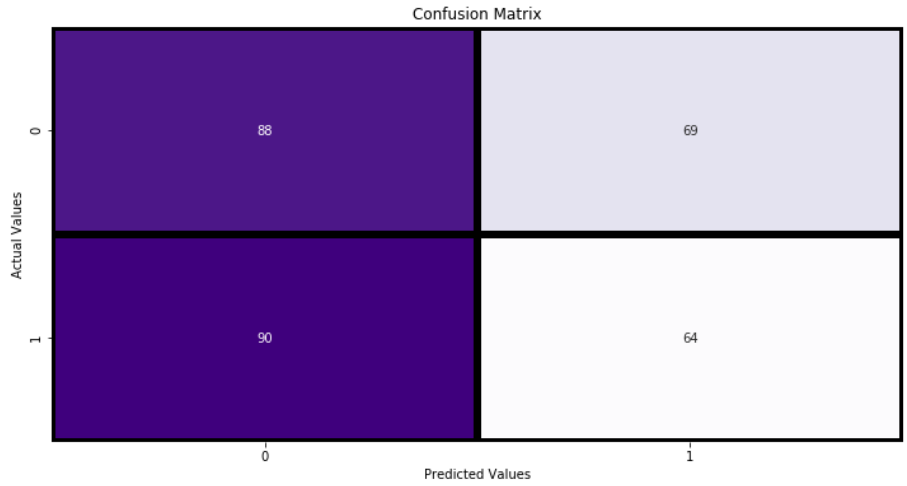
Decision Tree

در اینجا با استفاده از دستورات زیر به کمک روش درخت تصمیم به پیش بینی داده‌های تست میپردازیم:

Decision Tree for Classification

```
from sklearn.tree import DecisionTreeClassifier
classifierDT = DecisionTreeClassifier()
classifierDT.fit(X_train, y_train)
y_predDT = classifierDT.predict(X_test)
```

```
cm=confusion_matrix(y_test,y_predDT)
plt.figure(figsize=(12,6))
plt.title("Confusion Matrix")
sns.heatmap(cm, annot=True,fmt='d', cmap='Purples',linewidths=5,linecolor='black', cbar=False)
plt.ylabel("Actual Values")
plt.xlabel("Predicted Values")
plt.savefig('confusion_matrix.png')
```




```
print(classification_report(y_test, y_predDT))
```

	precision	recall	f1-score	support
0	0.49	0.54	0.52	157
1	0.48	0.44	0.46	154
micro avg	0.49	0.49	0.49	311
macro avg	0.49	0.49	0.49	311
weighted avg	0.49	0.49	0.49	311

```
test_acc = accuracy_score(y_test, y_predDT)
print("The Accuracy for Test Set is {}".format(test_acc*100))
```

```
The Accuracy for Test Set is 48.87459807073955
```

روش درخت تصمیم با دقت ۴۹٪ روزهای منفی را درست پیش بینی کرده است، یعنی ۸۵ روز از ۱۵۷ روز منفی را درست پیش بینی کرده است. همچین روش درخت تصمیم ۴۸٪ از روزهای مثبت را درست پیش بینی کرده است.

بطور کلی دقت روش درخت تصمیم ۴۸.۸٪ است. این دقت اصلا مناسب نیست و بسیار پایین است.

Random Forest

یک مدل یادگیری ماشینی دو نوع پارامتر دارد. پارامترهای نوع اول، پارامترهایی هستند که از طریق یک مدل یادگیری ماشینی یاد می گیرند، در حالی که پارامترهای نوع دوم، پارامترهایی هستند که به مدل یادگیری ماشینی منتقل می کنیم.

به طور معمول ما به صورت تصادفی مقدار این پارامترهای را تنظیم می کنیم و می بینیم که چه پارامترهایی بهترین عملکرد را دارند.

همچنین مقایسه عملکرد الگوریتم های مختلف با تنظیم تصادفی پارامترها آسان نیست زیرا ممکن است یک الگوریتم بهتر از دیگری با مجموعه پارامترهای مختلف عمل کند. و اگر پارامترها تغییر کنند، الگوریتم ممکن است بدتر از سایر الگوریتم ها عمل کند.

بنابراین، به جای انتخاب تصادفی مقادیر پارامترها، یک رویکرد بهتر توسعه الگوریتمی است که به طور خودکار بهترین پارامترها را برای یک مدل خاص پیدا کند. **Grid Search** یکی از این الگوریتم‌ها است.

به کد پایین با دقت نگاه کنید. در اینجا تابع **grid_param** را با چهار پارامتر **n_estimators، criterion ، bootstrap** و **max_depth** ایجاد می کنیم.

مقادیر پارامتری که می خواهیم امتحان کنیم در لیست مشخص می شوند. به عنوان مثال، در اسکریپت پایین می‌خواهیم پیدا کنیم که کدام مقدار (از ۲۰، ۵۰، ۱۵۰، ۲۰۰ و ۲۵۰) بالاترین دقت را دارد.

به طور مشابه، ما می خواهیم پیدا کنیم که کدام مقدار بالاترین عملکرد را برای پارامتر معیار دارد: **"gini"** یا **"آنتروپی"**؟ الگوریتم **Grid Search** اساساً تمام ترکیبات ممکن از مقادیر پارامتر را امتحان می کند و ترکیب را با بالاترین دقت برمی گرداند. به عنوان مثال، در مورد بالا الگوریتم ۹۶ ترکیب را بررسی می کند **(۹۶=4*2*2*6)**

الگوریتم **Grid Search** می تواند بسیار کند باشد، زیرا تعداد زیادی از ترکیبات برای آزمایش وجود دارد. علاوه بر این، اعتبار متقاطع زمان اجرا و پیچیدگی را افزایش می دهد.

Random Forest Classifier

```
from  sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```
classifierRF = RandomForestClassifier( random_state=0)
```

```
grid_param = {
   'n_estimators':[20,50,100,150,200,250],
   'criterion': ['gini', 'entropy'],
   'bootstrap': [True, False],
   'max_depth':[3,5,6,10]
}
```

```
gd_sr = GridSearchCV(estimator=classifierRF,
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1)
```

```
gd_sr.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
      estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
          max_depth=None, max_features='auto', max_leaf_nodes=None,
          min_impurity_decrease=0.0, min_impurity_split=None,
          min_samples_leaf=1, min_samples_split=2,
          min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
          oob_score=False, random_state=0, verbose=0, warm_start=False),
      fit_params=None, iid='warn', n_jobs=-1,
      param_grid={'n_estimators': [20, 50, 100, 150, 200, 250], 'criterion': ['gini', 'entropy'], 'bootstrap': [True, False], 'max_depth': [3, 5, 6, 1
0]},
      pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
      scoring='accuracy', verbose=0)
```

در زیر لیست بهترین پارامترها را برای اجرای جنگل تصادفی مشاهده میکنیم :

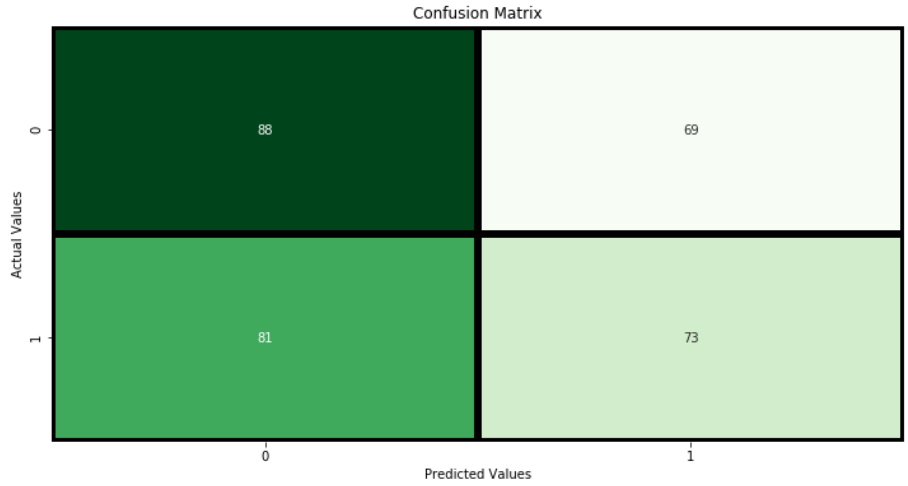
```
best_parameters = gd_sr.best_params_
print(best_parameters)
```

```
{'bootstrap': True, 'criterion': 'entropy', 'max_depth': 5, 'n_estimators': 150}
```

Predict : RandomForest

```
y_predRF = gd_sr.predict(X_test)
```

```
cm=confusion_matrix(y_test,y_predRF)
plt.figure(figsize=(12,6))
plt.title("Confusion Matrix")
sns.heatmap(cm, annot=True,fmt='d', cmap='Greens',linewidths=5,linecolor='black', cbar=False)
plt.ylabel("Actual Values")
plt.xlabel("Predicted Values")
plt.savefig('confusion_matrix.png')
```




```
test_acc = accuracy_score(y_test, y_predRF)
print("The Accuracy for Test Set is {}".format(test_acc*100))
```

```
The Accuracy for Test Set is 51.76848874598071
```

```
print(classification_report(y_test, y_predRF))
```

```
              precision    recall  f1-score   support

           0       0.52      0.56      0.54       157
           1       0.51      0.47      0.49       154

   micro avg       0.52      0.52      0.52       311
   macro avg       0.52      0.52      0.52       311
weighted avg       0.52      0.52      0.52       311
```

وقتی که به کمک بهترین پارامترهایی که بدست آوریدم جنگل تصادفی را اجرا میکنیم به دقت حدودا ۵۲٪ میرسیم.

دقت روش جنگل تصادفی برای پیش بینی روزهای منفی ۵۶٪ است و برای روزهای مثبت ۴۷٪ است.

SVM

ما داده ها را به مجموعه های آموزشی و آزمایشی تقسیم کرده ایم. اکنون زمان آموزش **SVM** بر روی داده های آموزشی است.

Scikit-Learn شامل کتابخانه **svm** است که شامل کلاس های داخلی برای الگوریتم های مختلف **SVM** است.

از آنجایی که قرار است یک کار طبقه بندی را انجام دهیم، از کلاس طبقه بندی کننده بردار پشتیبانی استفاده می کنیم که به صورت **SVC** در کتابخانه **svm Scikit-Learn** نوشته شده است.

این کلاس یک پارامتر دارد که نوع کرنل است. انتخاب نوع کرنل خیلی اهمیت دارد.

ما در اینجا از سه کرنل مختلف یعنی کرنل خطی، **Gaussian** و **sigmoid** استفاده میکنیم.

کرنل "خطی" فقط می توانند داده های قابل جداسازی خطی را طبقه بندی کنند.

با این حال، در مورد داده های غیرخطی قابل تفکیک، یک خط مستقیم نمی تواند به عنوان مرز تصمیم استفاده شود.

در مورد داده های غیرخطی قابل تفکیک، نمی توان از الگوریتم ساده **SVM** استفاده کرد. بلکه از نسخه اصلاح شده **SVM** به نام **Kernel SVM** استفاده می شود. برای مثال میتوان از کرنل "سیگموید" یا کرنل "گوسین" که کرنل‌های غیرخطی هستند استفاده کرد.

ما در اینجا به کمک الگوریتم **Grid Search** به انتخاب بهترین نوع کرنل و بقیه پارامترهای روش بردار پشتیبان میپردازیم .

Support Vector Machines (SVM) classifiers

```
from sklearn.svm import SVC
from sklearn import svm
```

Choose the best parameter for SVM and Training the Algorithm

```
classifierSVM = svm.SVC()
# defining parameter range
param_grid_svm = {'C': [0.001, 0.01, 0.05, 0.1, 0.5, 1.0, 10.0, 100],
          'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
          'gamma':['scale', 'auto'],
          'kernel': ['linear','rbf','sigmoid']}

gridSVM = GridSearchCV(estimator=classifierSVM,
                param_grid = param_grid_svm,
                scoring='accuracy',
                cv=10,
                refit=True,
                n_jobs=-1)

# fitting the model for grid search
gridSVM.fit(X_train, y_train)

# print best parameter after tuning
print(gridSVM.best_params_)
grid_predictions = gridSVM.predict(X_test)
```

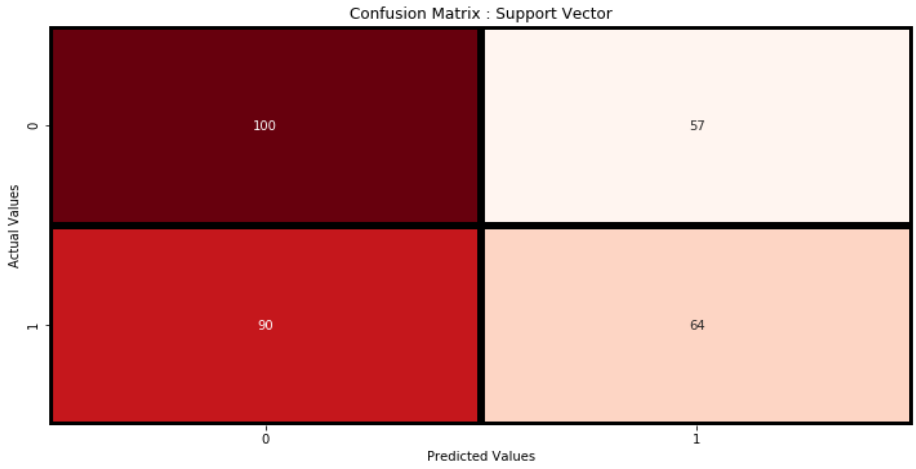
```
{'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf'}
```

در بالا بهترین پارامترهای مدل و همچنین بهترین کرنل مشخص شده است.

مدل خود را بر اساس این پارامترها آموزش میدهیم و به پیش بینی میپردازیم

print classification report : Support Vector

```
cm=confusion_matrix(y_test,grid_predictions)
plt.figure(figsize=(12,6))
plt.title("Confusion Matrix : Support Vector")
sns.heatmap(cm, annot=True,fmt='d', cmap='Reds',linewidths=5,linecolor='black',cbar=False)
plt.ylabel("Actual Values")
plt.xlabel("Predicted Values")
plt.savefig('confusion_matrix.png')
```




```
test_acc = accuracy_score(y_test, grid_predictions)
print("The Accuracy for Test Set is {}".format(test_acc*100))
```

The Accuracy for Test Set is 52.73311897106109

```
print(classification_report(y_test, grid_predictions))
```

	precision	recall	f1-score	support
0	0.53	0.64	0.58	157
1	0.53	0.42	0.47	154
micro avg	0.53	0.53	0.53	311
macro avg	0.53	0.53	0.52	311
weighted avg	0.53	0.53	0.52	311

مشاهده میشود که SVM دقتی ۵۲ درصدی دارد ولی نکته قابل توجه این است که SVM توانایی بسیار خوبی در پیش بینی روزهای منفی دارد، به گونه‌ای که روزهای منفی سهام را ۶۴٪ درست پیش بینی میکند.

ولی از طرفی روزهای مثبت قیت سهام را با دقتی ۴۲ درصدی درست پیش بینی میکند.

پیشنهاد میشود که برای انجام معامله فقط برای معاملات SHORT از svm استفاده کنید.

نتیجه‌گیری

Results table

```
index= ['KNN', 'DecisionTree', 'RandomForest', 'SVM']
cols = ['Accuracy for Test Set ', 'Accuracy for  Predicting 0', 'Accuracy for  Predicting 1']


df = pd.DataFrame(np.array([  [round(accuracy_score(y_test, y_predKNN)*100,2),classification_report(y_test, y_predKNN)[84:88]
                              ,(classification_report(y_test, y_predKNN)[138:143])]
                             ,[ round(accuracy_score(y_test, y_predDT)*100,2),classification_report(y_test, y_predDT)[84:88],
                               classification_report(y_test, y_predDT)[138:143]]
                            , [ round(accuracy_score(y_test, y_predRF)*100,2),classification_report(y_test, y_predRF)[84:88],
                              classification_report(y_test, y_predRF)[138:143]]
                            ,  [ round(accuracy_score(y_test, grid_predictions)*100,2),classification_report(y_test, grid_predictions)[84:88],
                              classification_report(y_test, grid_predictions)[138:143]]
                        ]),index=index, columns=cols)
df
```

	Accuracy for Test Set	Accuracy for Predicting 0	Accuracy for Predicting 1
KNN	55.63	0.59	0.52
DecisionTree	48.87	0.56	0.42
RandomForest	51.77	0.56	0.47
SVM	52.73	0.64	0.42

در جدول فوق میزان دقت روش‌هایی که در این پروژه استفاده کردیم را برای پیش بینی داده‌های تست مشاهده میکنیم.

برای تمام مجموعه داده‌های موجود در دیتای تست بالاترین دقت مربوط به الگوریتم **KNN** است با دقت ۵۵.۶ درصدی و پایین ترین دقت مربوط به روش درخت تصمیم با دقت۴۸.۸ درصد است.

مشاهده میشود الگوریتم **SVM** با دقت نسبتا خوبی میتواند روزهایی منفی قیمت سهام را پیش بینی کند، پس برای انجام معامله فروش یا به اصطلاح **short** میتوان از **svm** کمک گرفت همچنین ما اگر معامله خریدی باز کنیم یا به اصطلاح معامله **Long** داشته باشیم از روش **svm** میتوان برا بسته معامله استفاده کرد.

در کل مشاهده میشود که روش‌هایی که استفاده کردیم جهت منفی قیمت سهام را خیلی بهتر از جهت مثبت سهام پیش بینی میکنند.

دقت پیش بینی روزهای مثبت سهام در تمام روش‌ها به جز روش **knn** پایین تر از ۵۰٪ است که این یعنی اگر به کمک پرتاب سکه برای باز کردن معامله **Long** (خرید سهام) اقدام کنیم ضرر کمتری میکنم.

بطور کلی پیشنهاد میشود که از یک الگوریتم برای معامله کردن استفاده نشود، یک راه ساده این است که از تمام روش‌ها برای معامله استفاده کنیم یعنی اگر برای یک روز خاص از ۴ الگوریتم ۳ الگوریتم پیشنهاد معامله **Short** دادند ما وارد معامله بشویم.

البته که برای انجام معامله باید پارامترهای مختلفی را در نظر گرفت که خارج از بحث این پروژه است.

منابع

تمام خروجی‌های فوق به همراه کد پایتون در فایل proje data mining **قرار دارند.**

منبع بسیاری از کدها سایت https://stackabuse.com/ است.