- دریافت فایل داده ها (data set)

یک فایل csv که دارای ۱۵ ستون میباشد که به ترتیب از چپ به راست اطلاعات زیر را شامل میشوند:

- ✓ تاریخ
- ✓ نام نماد
- ✓ نوع امنیت
- ✓ قیمت لحظه بسته شدن روز قبل
- ✓ قیمت بازگشایی روز
- ✓ بیشترین قیمت روز
- ✓ کمترین قیمت روز
- ✓ قیمت آخرین معامله
- ✓ قیمت لحظه بسته شدن
- ✓ حجم وزنی میانگین قیمت
- ✓ حجم
- ✓ حجم معاملات
- ✓ معاملات (خالی میباشد)
- ✓ حجم تحویل
- ✓ درصد تحویل

- ذخیره کردن اطلاعات هر ستون در یک لیست و نمایش نمودارهای قیمت بازگشایی ، قیمت آخرین معامله و قیمت هنگام بسته شدن (open, last, close) برای شفاف سازی (visualization) داده ها.

```
#----------------------read data from file--------------------------
data = pandas.read_csv('ADANIPORTS.csv')

date, lastClose, open, high, low, last, close, volume = data['Date'], data['Prev Close'], data['Open'], data['High'], data['Low'], data['Last'], data['Close'], data['Volume']

#----------------------normalize values--------------------------
lastClose = normalize(lastClose)
open = normalize(open)
high = normalize(high)
low = normalize(low)
last = normalize(last)
close = normalize(close)
volume = normalize(volume)

#----------------------visualization--------------------------
plt.subplot(1,3,1)
plt.plot(open)
plt.ylabel('price')
plt.title('open')

plt.subplot(1,3,2)
plt.plot(last)
plt.title('last')

plt.subplot(1,3,3)
plt.plot(close)
plt.title('close')

plt.show()
```

برای آکاهی و فهم نتوع داده ها از شفاف سازی استفاده میشوند که باعث ایجاد دید بهتر به داده ها میشود.

- نرمال سازی داده ها برای تبدیل محدوده ی همه داده ها به مقداری بین 0 و 1 که در این صورت داده های بسیار بزرگ یا بسیار کوچک نسبت به بقیه داده ها ، باعث عملکرد غلط برنامه نمیشود و همه داده ها در یک محدوده قرار میگیرند. فرمول مورد استفاده برای نرمال سازی به صورت زیر میباشد:

$$X_{normal} = (X - X_{min}) / (X_{max} - X_{min})$$

- در متد com که به عنوان ورودی یک لیست میگیرد که شامل ویژگی های (بالاترین قیمت، پایین ترین قیمت، آخرین معامله، قیمت بسته شدن، نسبت حجم به کل معاملات و نسبت قیمت بسته شدن به آخرین معامله دخیره شده اند و داخل این متد همه ترکیب های ممکن از feature ها را تشکیل میدهیم و به عنوان یک لیست بر میگرداند و خروجی دوم آن یک رشته میباشد که نشان دهنده ترتیب feature های موجود در ایندکس متناظر میباشد.

```python
#-----------------------function for combination of lists---------------------------
def comb(lst:list):
    allFeatures = []
    combs = []
    for i in range(len(lst)):
        l = list(combinations(lst, i+1))
        for j in range(len(l)):
            tmp = []
            tmpStr = ''
            for k in range(i+1):
                indexList = l[j][k]
                if k == 0:
                    for m in range(len(indexList)):
                        tmp.append([indexList[m]])
                else:
                    for m in range(len(indexList)):
                        tmp[m].append(indexList[m])
                tmpStr += str(lst.index(indexList)) + ' '
            tmp.pop(-1)
            allFeatures.append(np.array(tmp))
            combs.append(tmpStr)
    return allFeatures, combs
```

- متد createModel که برای تبدیل داده ها به دو دسته train و test که روی داده های train به کامپیوتر آموزش میدهیم و روی داده های test چک میکنیم که آموزش ما چه نتیجه ای داشته و برنامه ما چه مقدار از داده ها را توانسته درست تشخیص دهد. برای این آموزش از الگوریتم KNN استفاده میکنیم که این الگوریتم به این صورت عمل میکند که k داده نزدیک داده اضافه شده را بررسی میکند و تعداد هرکدام بیشتر بود به آن گروه اضافه میشود(در این پروژه مقدار Positive یا Negative) میگیرد. در این متد پس از اجرای الگوریتم KNN و پیش بینی، آن را در یک dictionary ذخیره میکند و چاپ میکند و در آخر بیشترین درصد پیش بینی و ترتیب featureهای انتخابی را چاپ میکند و نمودار آنرا رسم میکند که در نمودار نقاط آبی پیش بینی درست هستند و نقاط قرمز نقاطی هستند ک به اشتباه پیش بینی شده اند.

```
#----------------------function for create model--------------------------
def createModel(allFeatures, y, combs):
    Max = -1
    bestFeature = ''
    predMax = -1
    predTmp = ''
    featureMax = ''
    j = 0
    for feature in allFeatures:
        X_train, X_test, y_train, y_test = train_test_split(feature, y, test_size=0.09)
        print(combs[j], ':')

        scores = {}
        for i in range(1,31):
            knn = KNeighborsClassifier(n_neighbors=i)
            knn.fit(X_train, y_train)
            pred = knn.predict(X_test)

            scores[i] = metrics.accuracy_score(y_test, pred)
            if predMax < scores[i]:
                predMax = scores[i]
                predTmp = pred

        Keymax = max(scores, key=scores.get)
        print(str(Keymax),':',scores[Keymax])
        if scores[Keymax] > Max:
            Max = scores[Keymax]
            bestFeature = combs[j]
            featureMax = y_test
        print('======================')
        j += 1
    print('number of all possible features:', len(allFeatures))
    print('best fatures:', bestFeature)
    print('max of all:', Max)

    for i in range(len(predTmp)):
        if predTmp[i] == featureMax[i]:
            plt.plot(featureMax[i], 'bo', markersize=1)
        else:
            plt.plot(predTmp[i], 'ro', markersize=1)
```