5ML PROJESİ

Hazırlayan: Emre Erol

Pozisyon: Stajyer

GİRİŞ

Bu projede yer alan uygulama başlıkları: Lineer Regresyon, Classification, Clustering, Association Rule ve Anomaly Detection olmaktadır. Her bir problem hem Jupiter arayüzü hem de Knime arayüzü temelli karşılanmaktadır. İki arayüz arasındaki sonuçlar değerlendirilmiş ve arasındaki farklar özelinde yorumlar yapılmıştır.

ÖZET

Proje için Covid-19 temelli sağlık sektörüne hizmet eden bir veri seti uygun görülmüş ve ilk olarak anomaly detection başlığı ile başlanıp box plot ve istatistiki olarak aykırı değer özelinde oluşturulan formüller ile çeşitli sütunlar üzerinde aykırı değer tahmini gerçekleştirilmiştir. Ardından farklı özellikler üzerinde Lineer Regresyon modeli eğitilmiş ve hata oranı değerlendirilmiştir. Daha sonrasında veri üzerinden çeşitli aggregate fonksiyonları yardımıyla sınıflandırma temeline uygun bir hale sokulmuş ve gerekli binning işlemleri uygulandıktan sonra classification yani sınıflandırma modeli eğitilmiş, hemen ardından ise oluşturulan veri seti üzerinden kümeleme modeli ile devam edilmiştir. Son olarak Association Rule ile yöntemleriyle bir kural çıkartılmış ve confidence değerleri özelinde ilginç bilgiler edinilmiştir.

1. ANOMALY DETECTION

Covid-19 veri setinde yer alan sütunlar arasından *Total_Deaths* üzerinde bir anomaly detection işlemi gerçekleştirilmesi uygun görülmüştür. *Total_Deaths* ile ilerlenirken iki farklı yöntem kullanılmıştır. Bunlardan ilki *Death_Momentum* diğeri ise değiştirilmemiş *Total_Deaths* sütunudur. *Death_Momentum* sütunu ise **Şekil 1**'de görüldüğü üzere toplam ölümlerin bir önceki gündeki değerinden çıkartılması ve bunun aynı işlemi bir önceki haline uygulanarak bölünmesi sonucu elde edilmiştir

1.1. Death_Momentum Sütunun Oluşturulması

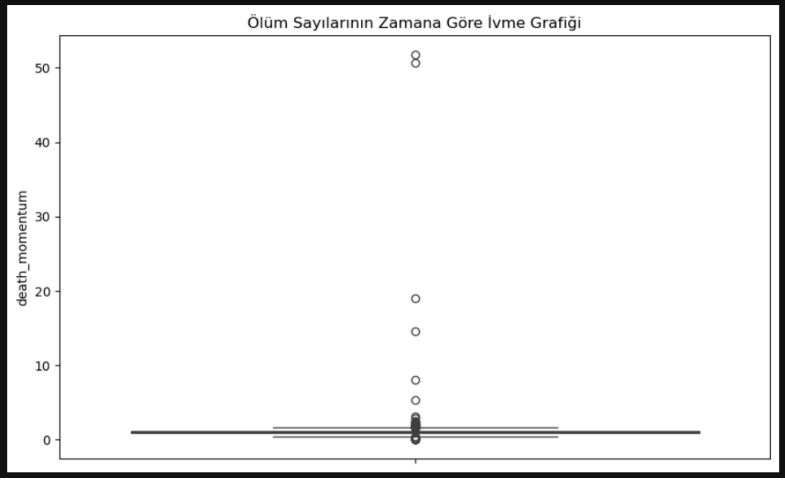
```
for i in range(2, len(dff_world['total_deaths'])):

    prev_deaths = dff_world.loc[i-1, 'total_deaths']
    prev_prev_deaths = dff_world.loc[i-2, 'total_deaths']

    death_momentum = (dff_world.loc[i, 'total_deaths'] - prev_deaths) / (prev_deaths - prev_prev_deaths)

    dff_world.loc[i, 'death_momentum'] = death_momentum
```

1.1.1. Şekil 1

Burada amaçlanan şey ise ölüm sayılarının birbirlerine bağımlı olarak değerlendirilmesi ve beklenmeyen ölüm sayılarının veri setine bağlı olarak bulunmasıdır. Daha sonrasında oluşturulan sütun üzerinde **Şekil 2**'de de görebileceğimiz üzere bir kutu grafiği oluşturulmuştur. *Death_Momentum* sütunumuzun satır sayısının fazla olması ve ivmeli bir büyüme gerçekleştirmesinden dolayı aşırı değer sayımız da fazla gözükmektedir.

1.2. Box Plot Grafiğinin Oluşturulması

```
[20]: plt.figure(figsize=(10,6))
      sns.boxplot(dff_world.death_momentum)
      plt.title('Ölüm Sayılarının Zamana Göre İvme Grafiği')
      # And with this box plot it is much easier to see that we have a bunch of outliers as a death count
      # Normally when we apply a linear method we want to have a stable momentum so that we have a good result

[20]: Text(0.5, 1.0, 'Ölüm Sayılarının Zamana Göre İvme Grafiği')
```



1.2.1. Şekil 2

Elde edilen bu grafikten sonra elimize geçen *"aykırı değerler var"* bilgisi özelinde gerekli istatistiki işlemler uygulanmış ve veri setinden aykırı değerler çekme işlemine geçilmiştir.

1.3. Aykırı Değerler Özelinde İstatistiki İşlemler

```
[21]: Q1 = dff_world['death_momentum'].quantile(0.25)
      Q3 = dff_world['death_momentum'].quantile(0.75)
      IQR = Q3 - Q1
      IQR

[21]: 0.2933175890676962

[22]: alt_sinir = Q1 - 1.5*IQR
      ust_sinir = Q3 + 1.5*IQR

[23]: ust_sinir

[23]: 1.5930675057814447

[24]: alt_sinir

[24]: 0.4197971495106597
```

1.3.1. Şekil 3

Şekil 3'te de gördüğünüz üzere alt ve üst değerler çeyrekler yardımıyla bulunmuş ve kutu grafiğinde işaretlenmiş olan üst ve alt bıyık (whisker)'ların sınırına ulaşılmıştır.

1.4. Aykırı Değerlerin Veri Setinden Çekilmesi

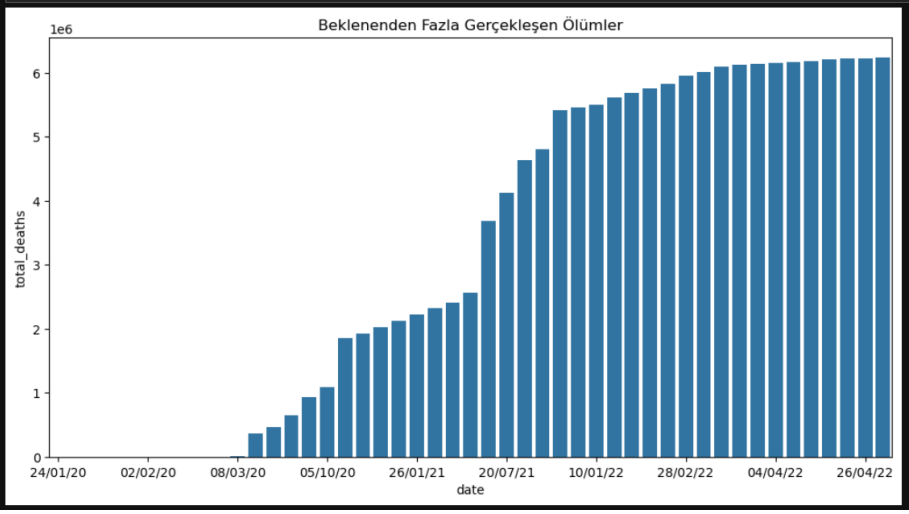
```
[25]: anormal_death_count1 = dff_world[(dff_world['death_momentum'] > ust_sinir)]
      anormal_death_count1.head()
```

[25]:

	iso_code	continent	location	date	total_cases	new_cases	total_deaths	new_deaths	total_vaccinations	people_vaccinated	people_fully_vaccinated	total_boosters	new_vaccinations
2	OWID_WRL	NaN	World	24/01/20	944.0	287.0	26.0	8.0	NaN	NaN	NaN	NaN	NaN
3	OWID_WRL	NaN	World	25/01/20	1437.0	493.0	42.0	16.0	NaN	NaN	NaN	NaN	NaN
5	OWID_WRL	NaN	World	27/01/20	2929.0	809.0	82.0	26.0	NaN	NaN	NaN	NaN	NaN
6	OWID_WRL	NaN	World	28/01/20	5580.0	2651.0	131.0	49.0	NaN	NaN	NaN	NaN	NaN
8	OWID_WRL	NaN	World	30/01/20	8237.0	2068.0	171.0	38.0	NaN	NaN	NaN	NaN	NaN

1.4.1. Şekil 4

```
[26]: plt.figure(figsize = (12,6))
      sns.barplot(x = anormal_death_count1.date, y = anormal_death_count1.total_deaths);
      plt.xticks(anormal_death_count1['date'][::5]);
      plt.title('Beklenenden Fazla Gerçekleşen Ölümler');
```



1.4.2. Şekil 5

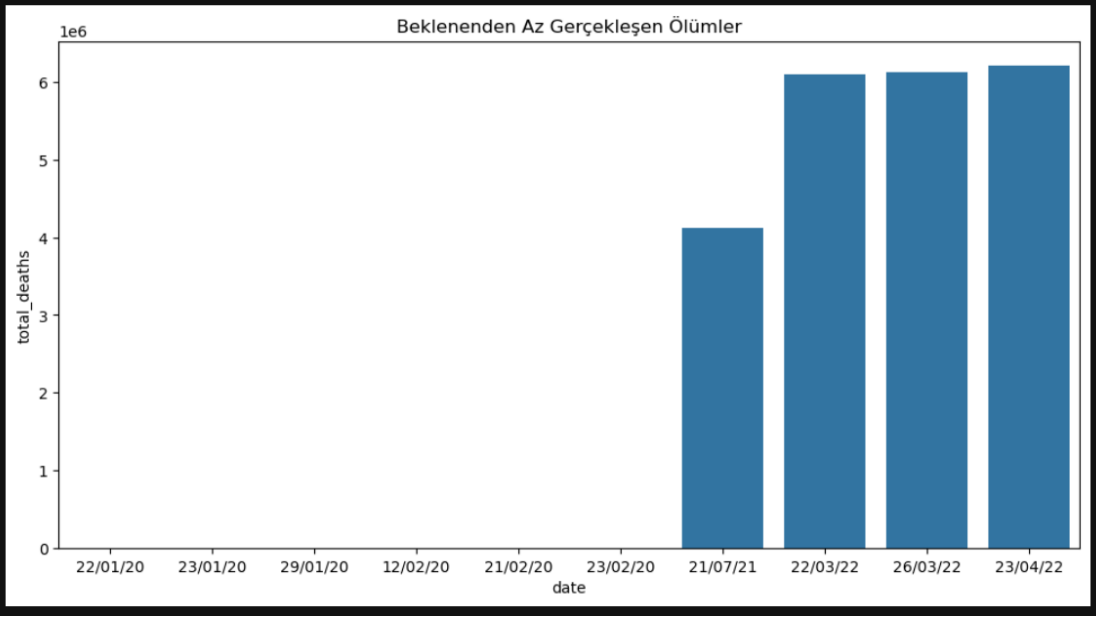
```
[27]: anormal_death_count2 = dff_world[(dff_world['death_momentum'] < alt_sinir)]
      anormal_death_count2
```

[27]:

	iso_code	continent	location	date	total_cases	new_cases	total_deaths	new_deaths	total_vaccinations	people_vaccinated	people_fully_vaccinated	total_boosters	new_vaccination
0	OWID_WRL	NaN	World	22/01/20	557.0	0.0	17.0	0.0	NaN	NaN	NaN	NaN	Na
1	OWID_WRL	NaN	World	23/01/20	657.0	100.0	18.0	1.0	NaN	NaN	NaN	NaN	Na
7	OWID_WRL	NaN	World	29/01/20	6169.0	589.0	133.0	2.0	NaN	NaN	NaN	NaN	Na
21	OWID_WRL	NaN	World	12/02/20	45232.0	418.0	1118.0	5.0	NaN	NaN	NaN	NaN	Na
30	OWID_WRL	NaN	World	21/02/20	76846.0	630.0	2252.0	4.0	NaN	NaN	NaN	NaN	Na
32	OWID_WRL	NaN	World	23/02/20	78990.0	382.0	2470.0	11.0	NaN	NaN	NaN	NaN	Na
546	OWID_WRL	NaN	World	21/07/21	192528904.0	555775.0	4127267.0	8660.0	3.782386e+09	2.104936e+09	8.285296e+08	6.091985e+06	32968235.(
790	OWID_WRL	NaN	World	22/03/22	474094730.0	1975437.0	6099469.0	4909.0	1.112885e+10	5.044412e+09	4.510733e+09	1.604364e+09	20545887.(
794	OWID_WRL	NaN	World	26/03/22	480816736.0	1356536.0	6122483.0	2929.0	1.118621e+10	5.059994e+09	4.525082e+09	1.629024e+09	18615290.(
822	OWID_WRL	NaN	World	23/04/22	509286465.0	516988.0	6217408.0	1424.0	1.152103e+10	5.125649e+09	4.634922e+09	1.830774e+09	13779771.(

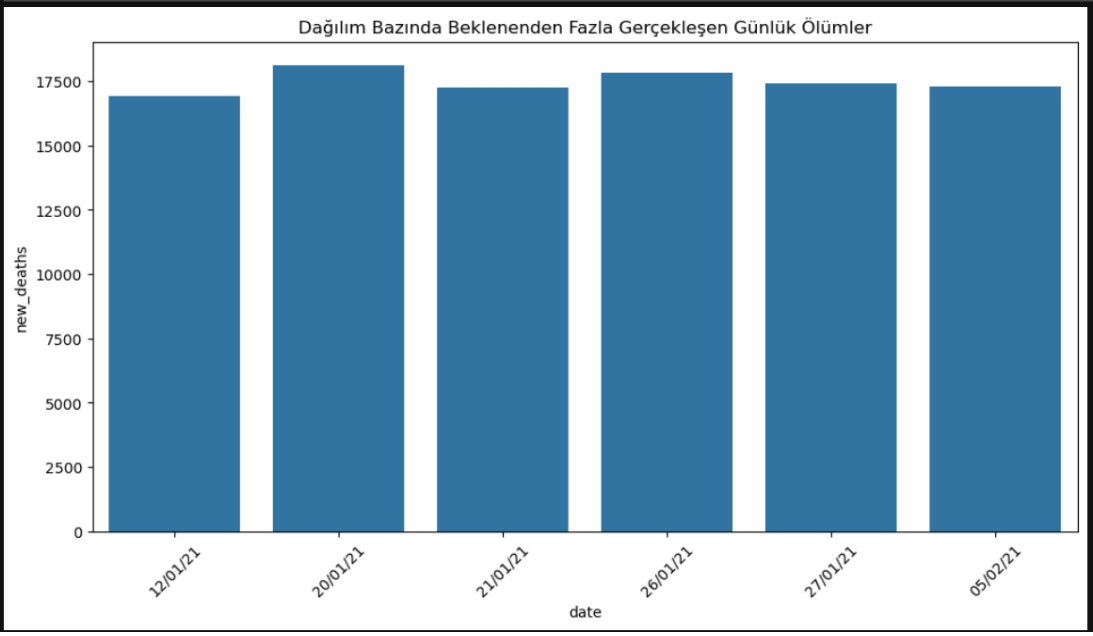
1.4.3. Şekil 6

```
[28]: plt.figure(figsize = (12,6))
      sns.barplot(x = anormal_death_count2.date, y = anormal_death_count2.total_deaths);
      plt.xticks(anormal_death_count2['date']);
      plt.title('Beklenenden Az Gerçekleşen Ölümler');
```



1.4.4. Şekil 7

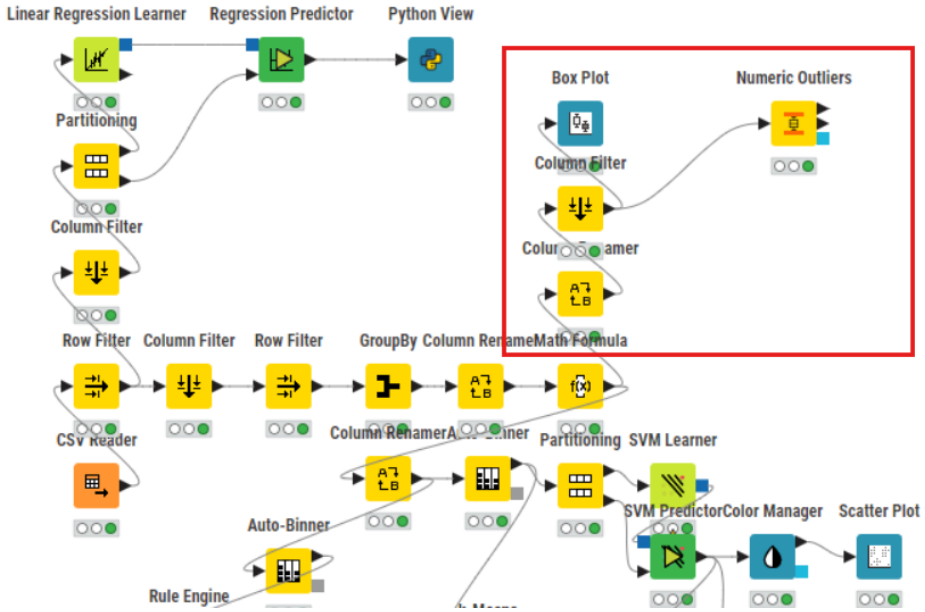
```
•[36]: plt.figure(figsize = (12,6))
       sns.barplot(x = new_outlier.date, y = new_outlier.new_deaths);
       plt.xticks(new_outlier['date'], rotation = 45);
       plt.title('Dağılım Bazında Beklenenden Fazla Gerçekleşen Günlük Ölümler');
```



1.4.5. Şekil 8

Şekil 4'te elde ettiğimiz veri setini barplot yardımıyla **Şekil 5**'te görebilmekteyiz. *Death_Momentum* sütununda beklenenden fazla ölümlere sahip toplam ölümler burada gözükmektedir. Ardından **Şekil 6**'da da gördüğümüz üzere alt sınıra geçilmiş ve beklenenden az gerçekleşen ölümler değerlendirilmiştir. Bu değerlere ise **Şekil 7** yardımıyla ulaşabilir ve inceleyebiliriz. Burada toplam ölüm değerlerinin gösterilmiş olmasından ötürü toplam ölümleri az olan baştaki satırlar gözükmemektedir. Fakat indeks bilgileri bizde olduğundan ötürü bu bilgiler ışığında yapılan aşı sayılarına, gerçekleşen yeni ölümlere ve gerçekleşen yeni vakalara da ulaşabiliriz. Son olarak **Şekil 8**'de ise günlük gerçekleşen ölümler değerlendirilmiş ve dağılım bazında değerlere ulaşılmıştır.

1.5. Knime Platformunda Anomali Tespiti



1.5.1. Şekil 9

Şekil 9'da kırmızı kenarlar özelinde belirtilen kısımda Knime üzerinde gerçekleştirilmiş olan anomali tespiti kısmı yer almaktadır. Knime üzerinde gerçekleştirdiğim anomali tespitinde ise *Death_Rate* sütunundan yani ölüm oranlarını içeren bir sütun üzerinden ilerledim. Bu sütuna ise *"total_deaths / total_cases"* formulü ile ulaştım. **Şekil 10**'da da görüldüğü üzere bir alt ve üst sınır *Numeric Outliers* düğümünden çıkmış bulundu.

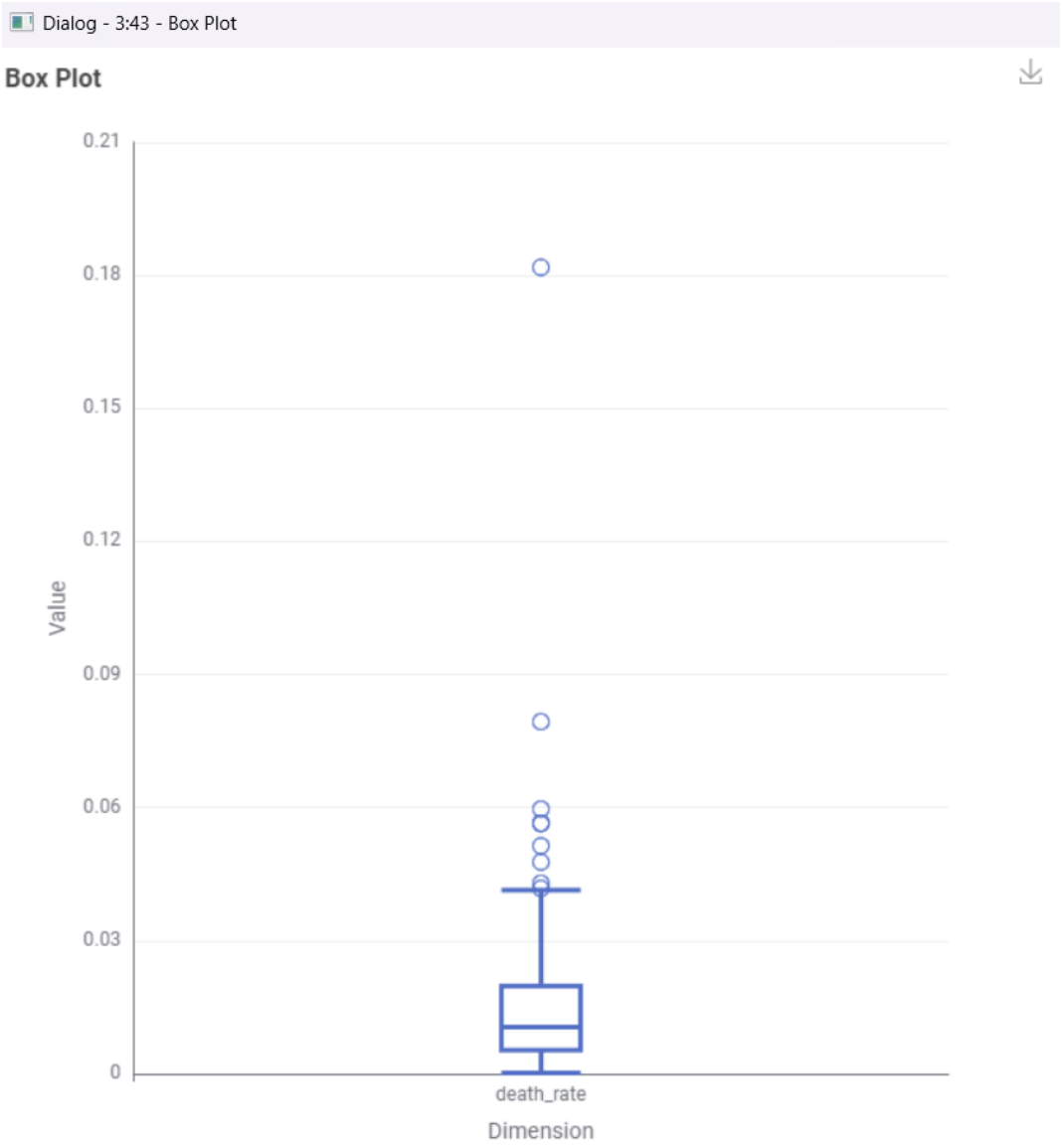
Summary (Table)

Rows: 1 | Columns: 5

#	RowID	Outlier column (String)	Member count (Number (integer))	Outlier count (Number (integer))	Lower bound (Number (double))	Upper bound (Number (double))
1	Row0	death_rate	209	9	-0.016	0.041

1.5.2. Şekil 10

Görüldüğü üzere sınır değerleri *-0.016* ve *0.041* olan iki sınır değerimiz bulunmuş oldu. *Death_Rate* sütununun kutu grafiğinde gösterimine ise **Şekil 11** yardımıyla ulaşabiliriz.



1.5.3. Şekil 11

Ve burada kutu grafiği yardımıyla aykırı değerlerin incelenmesini gerçekleştirebildik.

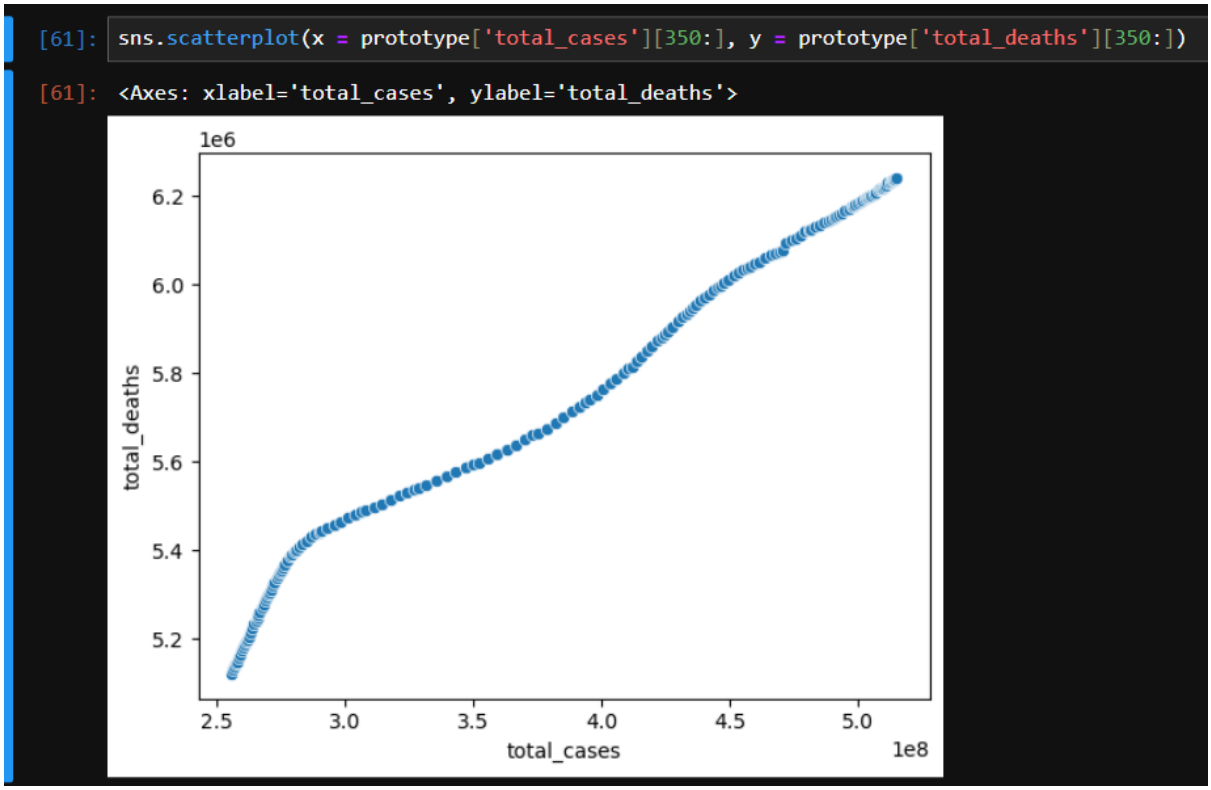
1.6. Sonuç

Hem Knime hem de Python özelinde gerekli sütunlar için aykırı değer tespiti ve analizi yapıldı. Bu sütunların bazıları default olarak tanımlanmışken bazıları ise çeşitli sütunlar yardımıyla çıkarıldı ve incelendi. Sonucunda faydalı bilgilere ve ilginç karşılaştırmalara rastlandı.

2. LINEAR REGRESSION

Lineer Regresyon için izlenen adımlar görece daha kısa ve nettir. *total_cases* ve *total_deaths* arasında bulunan ilişki lineer bir model yardımıyla incelenmiş ve gerekli doğruluk oranları elde edilmiştir. Lineer model fit edilmeden önce bir **Şekil 12**'de gözüktüğü üzere scatter plot yardımıyla iki sütun arasındaki ilişki gözlenmiş ve uygun görüldüğü üzere işleme geçilmiştir.

2.1. İki Özellik Arasındaki İlişkinin İncelenmesi



2.1.1. Şekil 12

Bu çıktı sonrasında lineer regresyon modeli için uygun olduğuna kanaat getirilmiş ve devam edilmiştir. Çıktının bu şekilde olmasının sebebi ise veri setinin zaman indeksli bir yapıya sahip olması dolayısıyla her bir değerin zamanla eş zamanlı olarak artmasıdır. Ardından hata kareler ortalamasına bakılmış ve varyans ile karşılaştırılarak doğruluk ölçütü gerçekleştirilmiştir. Varyansa kıyasla oldukça düşük kalan hata kareler değerimiz bize sonucumuzun **Şekil 13**'te de görüldüğü üzere gayet iyi olduğunu söylemektedir.

2.2. Modelin Fit Edilmesi ve Hata Oranlarının Değerlendirilmesi

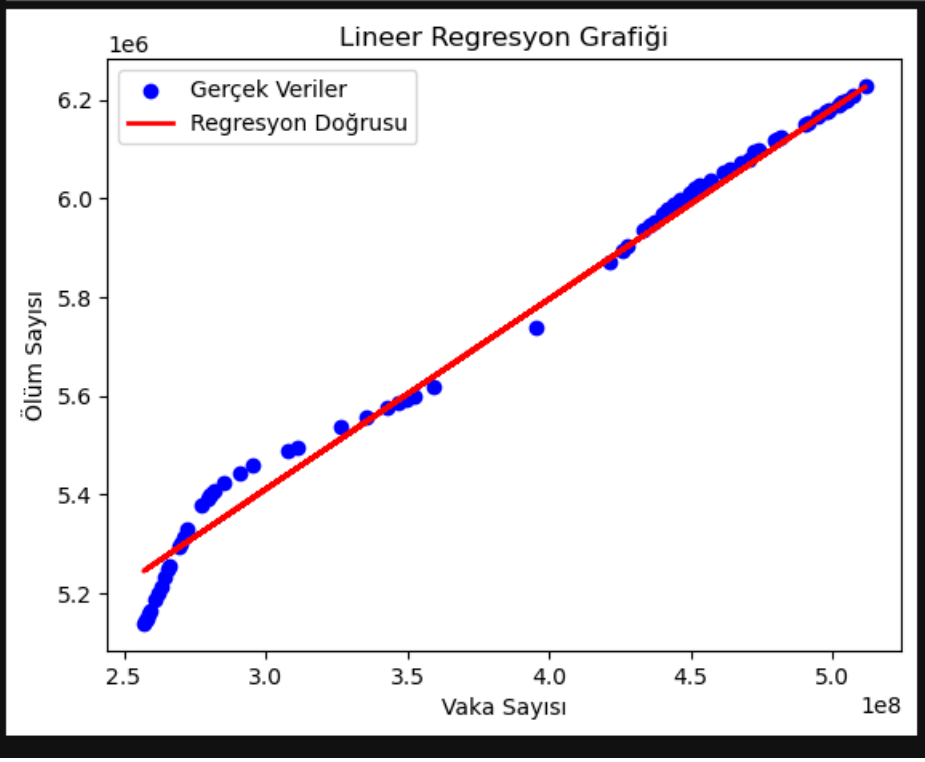
```
[65]: from sklearn.metrics import mean_squared_error
      ln = LinearRegression()
      model = ln.fit(X1_train, y1_train)
      y_pred = model.predict(X1_test)
      mean_squared_error(y1_test, y_pred)

[65]: 1637026202.2005844

[66]: variance = np.var(y1_test)
      variance

[66]: 139777438792.36765
```

Şekil 2.2.1. Şekil 13

Daha sonrasında elde edilen sonuç net bir izlenim kazanılması adına **Şekil 14**'te de görüldüğü üzere görselleştirilmiş ve yorumlanmıştır.

2.3. Modelin Görselleştirilmesi

```
[67]: X1_ = X1_test

      plt.scatter(X1_, y1_test, color='blue', label="Gerçek Veriler")

      plt.plot(X1_, y_pred, color='red', linewidth=2, label="Regresyon Doğrusu")


      plt.xlabel("Vaka Sayısı")
      plt.ylabel("Ölüm Sayısı")
      plt.title("Lineer Regresyon Grafiği")
      plt.legend()
      plt.show()
```

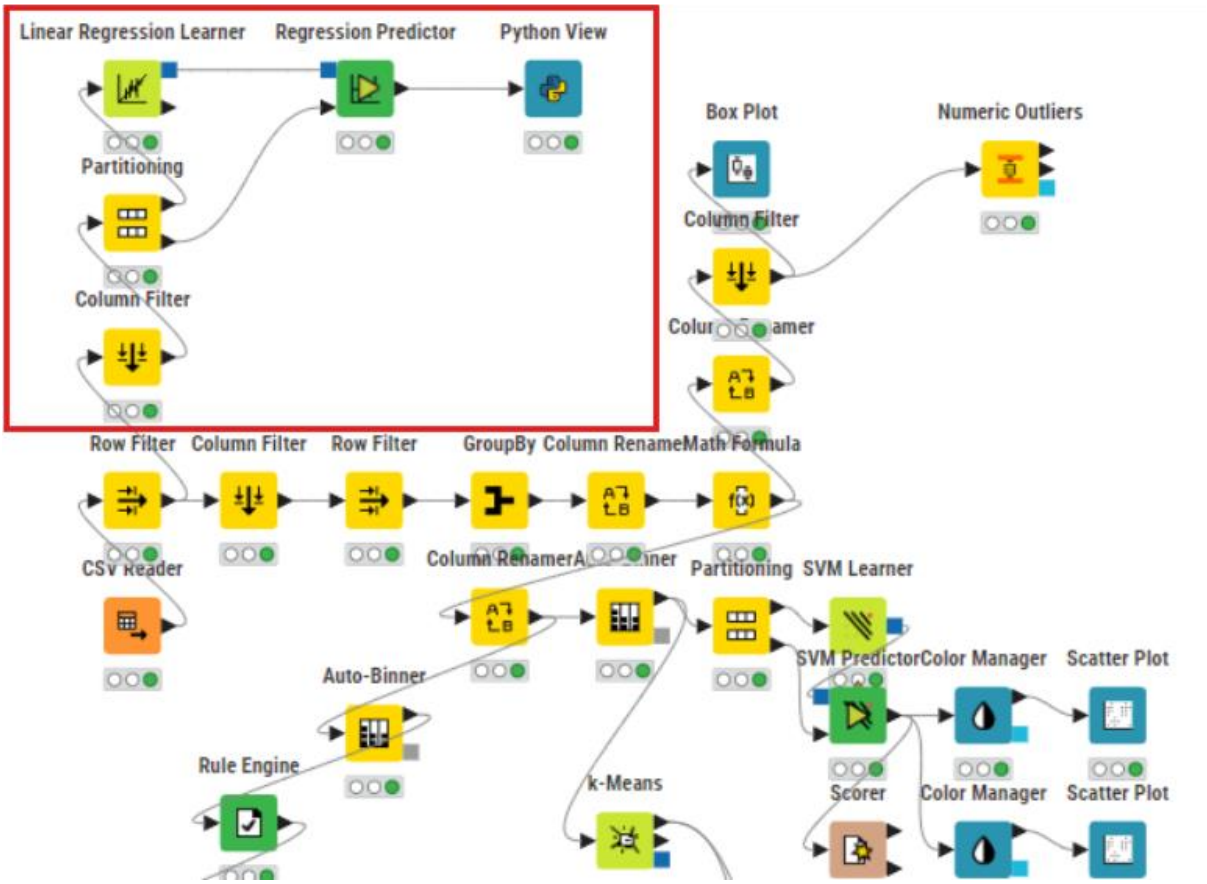


2.3.1. Şekil 14

Görüldüğü üzere kırmızı doğru etrafında şekillenen mavi noktalar modelimizin sorunsuz bir şekilde çalıştığını ve iyi yönde bir fit etme işleminin gerçekleştiğini bize aktarıyor.

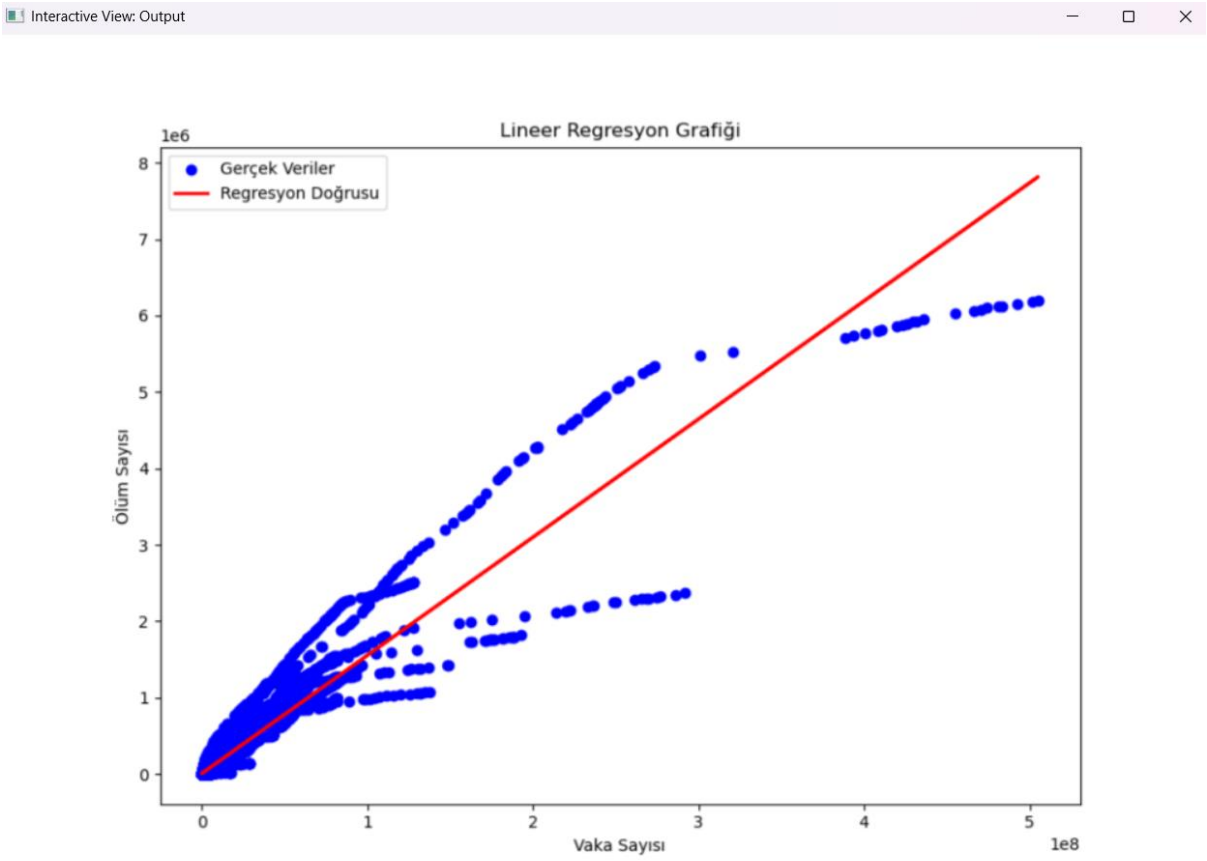
2.4. Knime ile Lineer Regresyon Fit Edilmesi

Knime ile Lineer regresyon modelinin kurulması yolunda görselleştirmeler adına birkaç sorun çıkmış ve bu sorun python snippetleriyle giderilmiştir. **Şekil 15** üzerinde de görebileceğimiz üzere Lineer regresyon modeli önce gerekli filter işlemlerinden geçen ve daha sonrasında test ve train setine ayrılan veri seti üzerinden eğitilmiş ve görselleştirilmesi yapılmıştır.



2.4.1. Şekil 15

Daha sonrasında Python View yardımıyla **Şekil 16**'da görüldüğü üzere çıktısı alınmış ve bitirilmiştir.



2.4.2. Şekil 16

2.5. Sonuç

Lineer regresyon modelini kullanmadan önce dağılımın incelenmesi ardından gerekli fit etme işlemlerinin gerçekleştirilmesi ve daha sonrasında görselleştirme işlemlerinin uygulanması modelin iki önemli özelliği sayılabilecek *total_cases* ve *total_deaths* sütunlarının arasındaki ilişkiyi daha net görmemizi sağlamıştır.

3. CLASSIFICATION

Elimizdeki veri seti üzerinde gerçekleştirilen sınıflandırma problemleri veri setine çeşitli aggregate ve groupby işlemleri sonrasında gerçekleşmiştir. Öncesinde *continent* ve *location* sütunları harici herhangi bir string değer olmamasından kaynaklı **Şekil 17**'de de görüldüğü üzere *"(total_deaths / total_cases) * 100"* ile oluşturulan risk_*rate* sütununa binning işlemi uygulanmış ve bu işlem sonucu çıkan sütun özelinde modeller eğitilmiştir.

3.1. risk_rate Sütununun İnşası

```
[93]: risk_rate = dff.groupby(['continent', 'location'])[['total_cases', 'total_deaths']].max().reset_index()
      risk_rate['risk_rate'] = (risk_rate['total_deaths'] / risk_rate['total_cases']) * 100
      risk_rate
```

[93]:

	continent	location	total_cases	total_deaths	risk_rate
0	Africa	Algeria	265782.0	6875.0	2.586706
1	Africa	Angola	99287.0	1900.0	1.913644
2	Africa	Benin	86394.0	163.0	0.188671
3	Africa	Botswana	305984.0	2688.0	0.878477
4	Africa	Burkina Faso	20865.0	383.0	1.835610
...	...	...	...	...	...
225	South America	Paraguay	649455.0	18870.0	2.905513
226	South America	Peru	3567171.0	212877.0	5.967670
227	South America	Suriname	79393.0	1328.0	1.672692
228	South America	Uruguay	899723.0	7210.0	0.801358
229	South America	Venezuela	522514.0	5709.0	1.092602

230 rows × 5 columns

3.1.1. Şekil 17

Groupby metodu ile *continent* ve *location* sütunlarına göre gruplama işlemi gerçekleştirilmiş ve daha sonrasında max değeri ile maksimum değerleri çekilerek aggregate işlemi gerçekleştirilmiştir. Daha sonrasında buradan çıkan çıktı *risk_rate* adlı bir dataframe'e aktarılmış ve binning işlemine **Şekil 18**'de görüldüğü üzere aktarılan dataframe üzerinden devam edilmiştir.

3.2. risk_rate Sütunu Üzerinde Binning İşlemi Gerçekleştirilmesi

```
[97]: risk_rate['risk_category'] = pd.qcut(risk_rate['risk_rate'], q=4, labels=['Low', 'Medium', 'High', 'Very High'])
      risk_rate
```

[97]:

	continent	location	total_cases	total_deaths	risk_rate	risk_category
0	Africa	Algeria	265782.0	6875.0	2.586706	Very High
1	Africa	Angola	99287.0	1900.0	1.913644	High
2	Africa	Benin	86394.0	163.0	0.188671	Low
3	Africa	Botswana	305984.0	2688.0	0.878477	Medium
4	Africa	Burkina Faso	20865.0	383.0	1.835610	High
...	...	...	...	...	...	...
225	South America	Paraguay	649455.0	18870.0	2.905513	Very High
226	South America	Peru	3567171.0	212877.0	5.967670	Very High
227	South America	Suriname	79393.0	1328.0	1.672692	High
228	South America	Uruguay	899723.0	7210.0	0.801358	Medium
229	South America	Venezuela	522514.0	5709.0	1.092602	High

230 rows × 6 columns

3.2.1. Şekil 18

Görüldüğü üzere gerçekleştirilen binning işlemleri qcut metodu ile yani çeyrekler ile belirlenen sınırları ve sınırların arasında kalan alanları baz alacak şekilde yapılmıştır. Bu aralıklara ise sırasıyla Low, Medium, High ve Very High şeklinde isimlendirmeler yapılmıştır. Daha sonrasında **Şekil 19**'da görüldüğü üzere veri seti üzerinde null değer kontrolü yapılmış ve tespit edilen null değerler kaldırılmıştır.

3.3. Elde Edilen DataFrame Üzerinde Null Değer Kontrolü

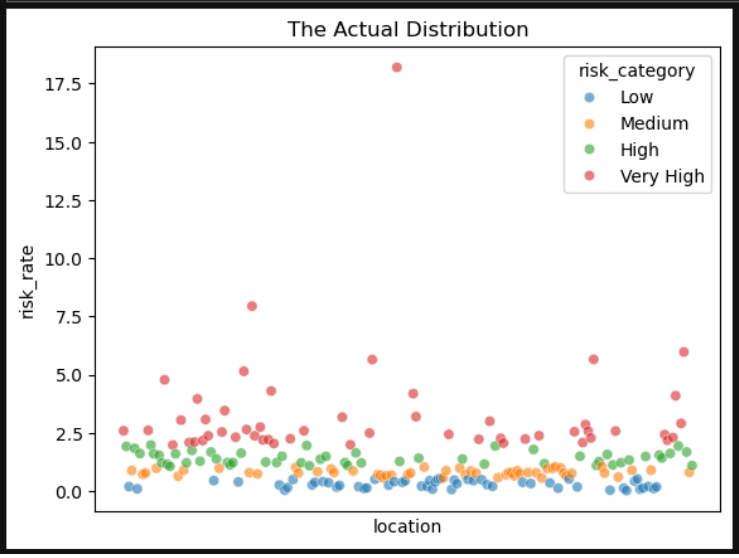
```
[98]: risk_rate.isnull().sum()

[98]: continent          0
      location           0
      total_cases       15
      total_deaths      21
      risk_rate         21
      risk_category     21
      dtype: int64

[99]: risk_rate = risk_rate.dropna(subset = ['total_deaths'])
```

3.3.1. Şekil 19

Yapılan testler sonucu 21 tane null değer tespit edilmiş ve bu eksikler *total_deaths* sütunu baz alınarak yok edilmiştir. Daha sonrasında **Şekil 20** ve **Şekil 21**'de görüleceği üzere elimizdeki veri seti görselleştirilmiş ve incelenmiştir.

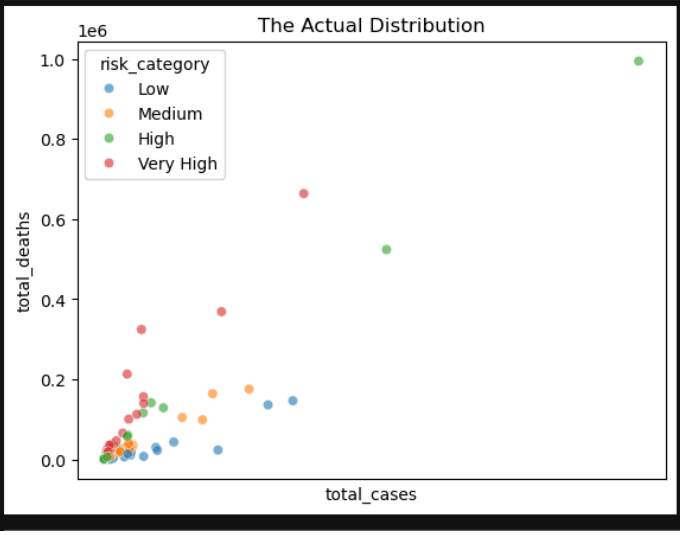
3.4. Elde Edilen DataFrame Üzerinde Görsel İncelemeler

```
[100]: sns.scatterplot(x = risk_rate.location, y = risk_rate.risk_rate, hue = risk_rate.risk_category, alpha=0.6)
       plt.xticks([])
       plt.title("The Actual Distribution")
       plt.show()
```



3.4.1. Şekil 20

```
[330]: sns.scatterplot(x = risk_rate.total_cases, y = risk_rate.total_deaths, hue = risk_rate.risk_category, alpha=0.6)
       plt.xticks([])
       plt.title("The Actual Distribution")
       plt.show()
```



3.4.2. Şekil 21

Elde edilen *total_deaths* ve *total_cases* sütunları her bir ülkeye göre en yüksek değerler olmasından kaynaklı elimizdeki ana veri setini zaman indeksli bir yapıdan çıkaran ve dünya bazında genel olarak riskleri değerlendirebilen bir yapı ile bizi karşılamaktadır. **Şekil 20**'de risk yüzdelerine göre, **Şekil 21**'de ise *total_deaths* ve *total_cases* arasındaki ilişkiye göre bir grafik bizi karşılamaktadır. **Şekil 21**'deki bu yapıyı, grafiğe grafiğin köşegeni olacak şekilde

bir çizgi koyduğumuzda yukarda kalanların High ve Very High olmasından anlayabiliriz. Herhangi bir kümelenmenin gerçekleşmemiş olmasından kaynaklı KNN gibi uzaklık tabanlı sınıflandırma algoritmalarının düşük sonuç vermesini beklerken SVC gibi vektörler aracılığı ile sağlanan sınıflandırma problemlerinin daha yüksek sonuç vermesini bekleriz.

3.5. Modellerin Denenmesi ve Değerlendirilmesi

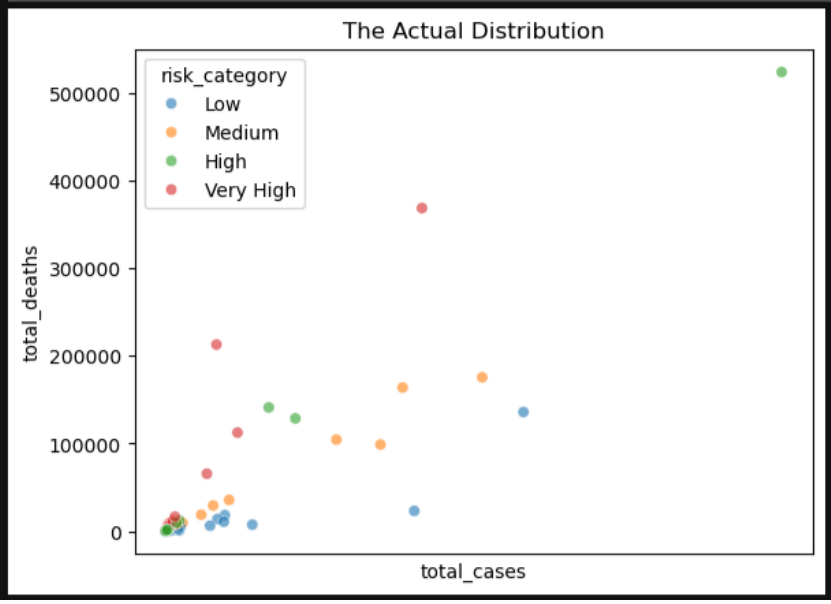
3.5.1. K Nearest Neighbor Modeli

```
[102]: knn1 = KNeighborsClassifier()

[103]: X1_c = risk_rate[['total_cases', 'total_deaths']]
       y1_c = risk_rate['risk_category']

[104]: Xc_train1, Xc_test1, yc_train1, yc_test1 = train_test_split(X1_c, y1_c, test_size=0.4, random_state=40)

[105]: knn_model1 = knn1.fit(Xc_train1, yc_train1)
       yc_pred1 = knn_model1.predict(Xc_test1)
       accuracy1 = accuracy_score(yc_test1, yc_pred1)

[106]: accuracy1

[106]: 0.38095238095238093

[107]: cv_scores1 = cross_val_score(knn1, X1_c, y1_c, cv=10, scoring='accuracy')

[108]: cv_scores1.mean()

[108]: 0.26761904761904765
```

3.5.1.1. Şekil 22

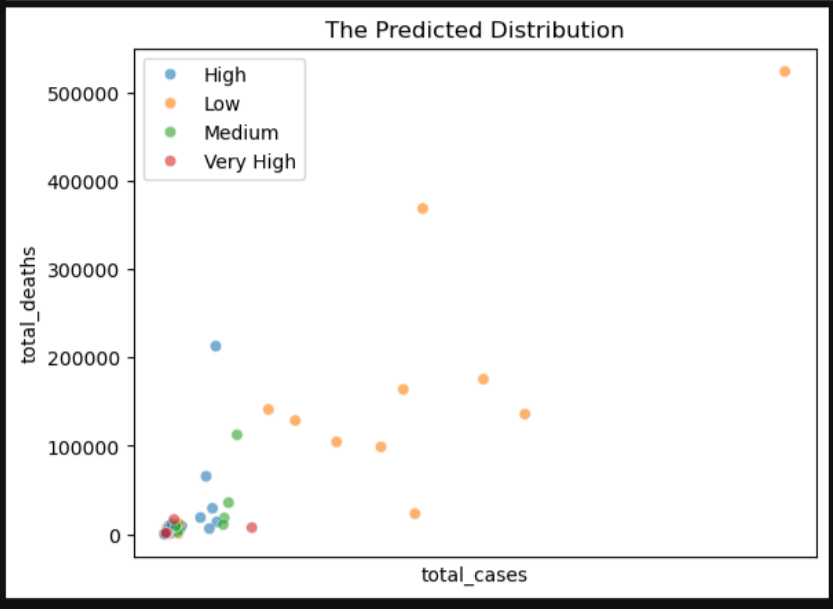
Şekil 22'de gözüktüğü gibi KNN modeli kötü denilebilecek bir accuracy oranı getirmiştir. Daha sonrasında bu değer cross validation metodu ile onaylanmıştır. Modelin çıktısının kötü olduğu bilinse bile görsellendirme yardımıyla modelin sonucu **Şekil 23** ve **Şekil 24**'te gözüktüğü üzere incelenmiştir.

```
[109]: sns.scatterplot(x = Xc_test1.total_cases, y = Xc_test1.total_deaths, hue = yc_test1, alpha=0.6)
       plt.xticks([])
       plt.title("The Actual Distribution")
       plt.show()
```



3.5.1.2. Şekil 23

```
[110]: sns.scatterplot(x = Xc_test1.total_cases, y = Xc_test1.total_deaths, hue = yc_pred1, alpha=0.6)
       plt.xticks([])
       plt.title("The Predicted Distribution")
       plt.show()
```



3.5.1.3. Şekil 24

"The Actual Distribution" yani gerçek dağılım ile *"The Predicted Distribution"* arasındaki farklar görseller yardımıyla daha net bir potaya çekilmiştir. KNN Modelinin tahmin ettiği dağılım görüldüğü üzere kümelenmeye daha müsait ve gruplaşmaya daha müsait bir yapı ile

bizi karşılamaktadır. Bu sayede modelimizin neden kötü bir sonuç verdiği anlaşılmış ve knn modelimizin tahmin ettiğimiz üzere kötü bir sonucu nasıl verdiği gözlemlenmiştir.

3.5.2. Decision Tree Modeli

```
[111]: from sklearn.tree import DecisionTreeClassifier
       dcs = DecisionTreeClassifier(max_depth=5)
       decision_model = dcs.fit(Xc_train1, yc_train1)

[112]: y_decision_pred = decision_model.predict(Xc_test1)

[113]: decision_accuracy = accuracy_score(yc_test1, y_decision_pred)

[114]: decision_accuracy

[114]: 0.5476190476190477

[115]: decision_cv_scores = cross_val_score(dcs, X1_c, y1_c, cv=10, scoring='accuracy')

[116]: decision_cv_scores.mean()

[116]: 0.41666666666666663
```

3.5.2.1. Şekil 25

Decision Tree modelimiz ise **Şekil 25**'te görüldüğü üzere kötü denilebilecek bir çıktı vermiş ve cross validation yöntemi ile doğrulanmıştır.

3.5.3. Random Forest Modeli

```
[117]: from sklearn.ensemble import RandomForestClassifier
       rfc = RandomForestClassifier(n_estimators=100)
       random_model = rfc.fit(Xc_train1, yc_train1)

[118]: y_random_pred = random_model.predict(Xc_test1)

[119]: random_accuracy = accuracy_score(yc_test1, y_random_pred)

[120]: random_accuracy

[120]: 0.6785714285714286

[121]: random_cv_scores = cross_val_score(rfc, X1_c, y1_c, cv=10, scoring='accuracy')

[122]: random_cv_scores.mean()

[122]: 0.7319047619047618
```

3.5.3.1. Şekil 26

Random Forest modelimiz Decision Tree'den beklendiği ve **Şekil 26**'da da görüldüğü üzere daha iyi sonuç vermiş ve diğer algoritmalardan farklı olarak cross validation yöntemi

uygulandığında daha yüksek doğruluk oranına çıkmıştır. Bunu sağlayan şey ise Decision Tree'den daha dengeli ve daha kapsamlı bir algoritmaya sahip olmasıdır.

3.5.4. XGBoost Modeli

```
[123]: from xgboost import XGBClassifier
       from sklearn.preprocessing import LabelEncoder
       le = LabelEncoder()
       xgb = XGBClassifier()

       y_encoded_train = le.fit_transform(yc_train1)
       xgb_model = xgb.fit(Xc_train1, y_encoded_train)

[124]: y_xgb_pred = xgb_model.predict(Xc_test1)

[125]: y_encoded_test = le.fit_transform(yc_test1)

[126]: xgb_accuracy = accuracy_score(y_encoded_test, y_xgb_pred)

[127]: xgb_accuracy

[127]: 0.6071428571428571

[128]: y_encoded = le.fit_transform(y1_c)

[129]: xgb_cv_scores = cross_val_score(xgb, X1_c, y_encoded, cv=10, scoring='accuracy')

[130]: xgb_cv_scores.mean()

[130]: 0.6845238095238095
```

3.5.4.1. Şekil 27

XGBoost modelimiz **Şekil 27**'de görüldüğü üzere fena olmayan bir doğruluk oranı ile bizi karşılasa da maalesef yeterli sonucu elde edememiştir. Random Forest ile benzer şekilde cross validation ile doğrulandığı zaman doğruluk oranı yükselmiştir. XGBoost modelimizin Decision Tree algoritmasından yine daha yüksek doğruluk oranı vermesini kendi temasını boostlama üzerine kurmasından bekleyebiliriz.

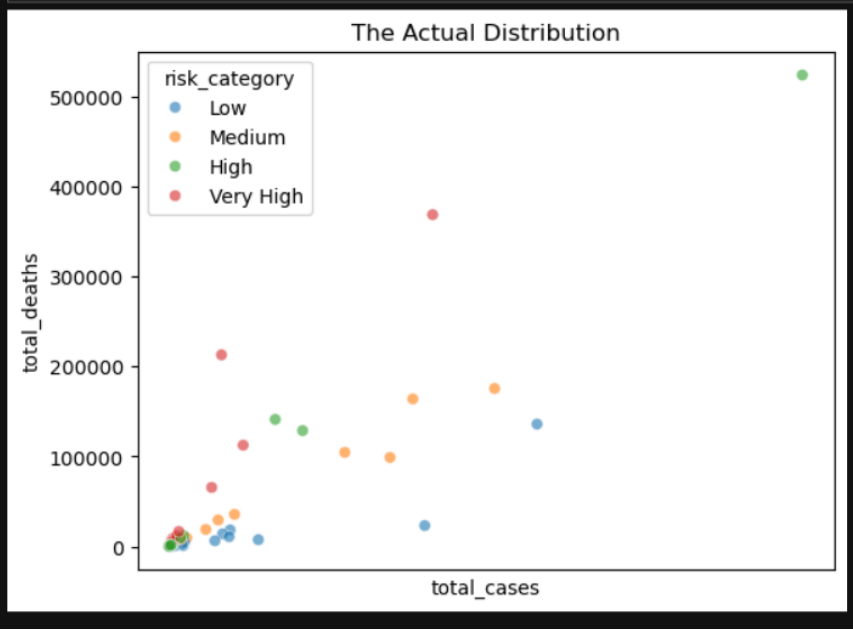
3.5.5. Support Vector Classifier Modeli

```
[131]: from sklearn.svm import SVC
       svc = SVC(kernel='linear')
       svc_model = svc.fit(Xc_train1, yc_train1)

[132]: y_svc_pred = svc_model.predict(Xc_test1)

[133]: svc_accuracy = accuracy_score(yc_test1, y_svc_pred)

[134]: svc_accuracy

[134]: 0.9761904761904762

[135]: svc_cv_scores = cross_val_score(svc, X1_c, y1_c, cv=10, scoring='accuracy')

[136]: svc_cv_scores.mean()

[136…  0.980952380952381
```

3.5.5.1. Şekil 28

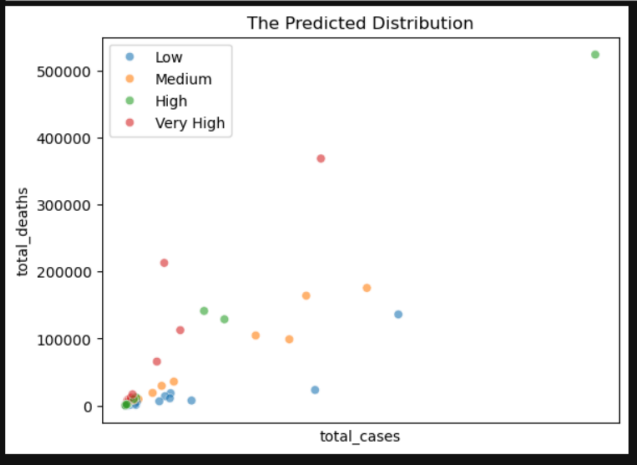
Şekil 28'de görüldüğü üzere SVC modelinde diğer modellere kıyasla çok daha yüksek miktarda doğruluk oranı elde edilmiş ve cross validation ile doğrulanmıştır. Yüksek doğruluk elde edilmesinin sebebi Support Vector Classifier algoritmasının vektörler yardımıyla öğrenmesi ve tahmin etmesidir. Modeli oluştururken *total_deaths* sütununu *total_cases* sütununa böldüğümüz için oluşan grafiğin de lineer olarak ayrılabilen bir yapıda olması SVC modelini burada kazanan kılmıştır. Modelin çıktısı ise **Şekil 29** ve **Şekil 30** yardımıyla gerçek veriler ile karşılaştırılarak incelenmiştir.

```
[137]: sns.scatterplot(x = Xc_test1.total_cases, y = Xc_test1.total_deaths, hue = yc_test1, alpha=0.6)
       plt.xticks([])
       plt.title("The Actual Distribution")
       plt.show()
```



3.5.5.2. Şekil 29

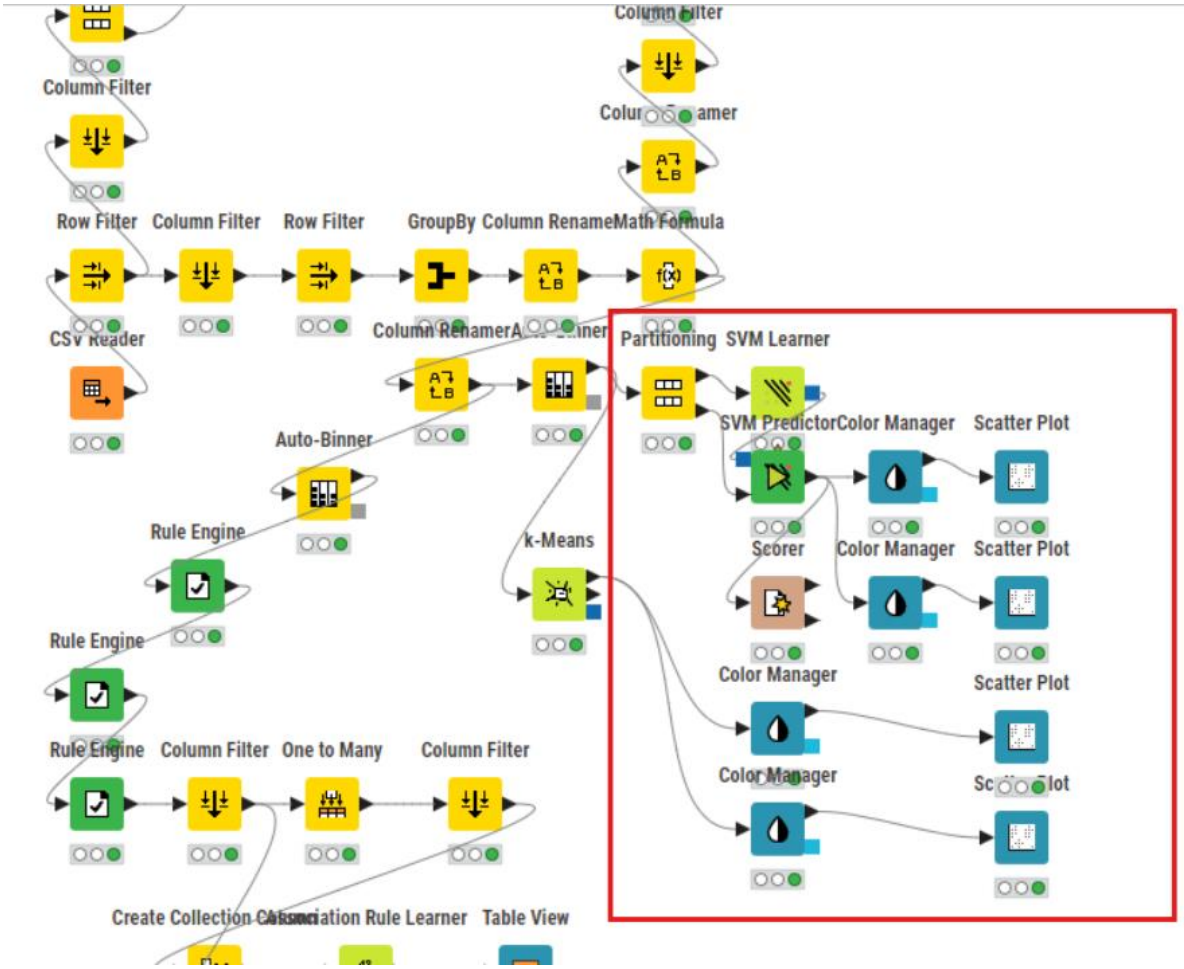
```
[138]: sns.scatterplot(x = Xc_test1.total_cases, y = Xc_test1.total_deaths, hue = y_svc_pred, alpha=0.6, hue_order=['Low', 'Medium', 'High
       plt.xticks([])
       plt.title("The Predicted Distribution")
       plt.show()
```



3.5.5.3. Şekil 30

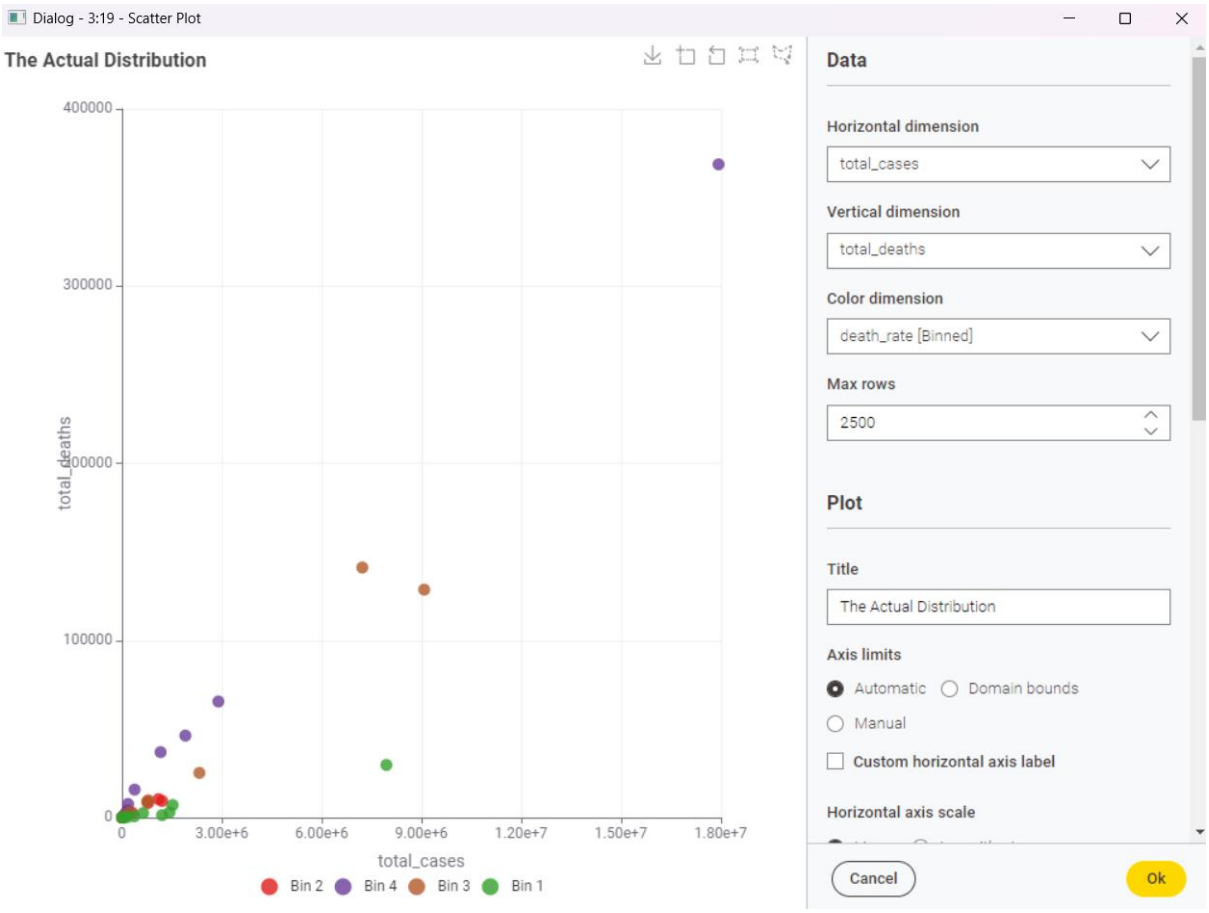
"The Actual Distribution" ile "The Predicted Distribution" arasında bir karşılaştırılma yapıldığı zaman arasındaki farkların oldukça az olduğu ve insan gözüyle fark edilmesinin gerçekten zor olduğu görülmektedir. Daha öncesinde de bahsedilmiş olan SVC modelinin vektörlerini burada göremesek bile varlığını sezgisel olarak fark etmek oldukça mümkündür. Veri seti tahminlerinin lineer bir çizgi aracılığıyla ayrıldığı ise sonuçlarda açıkça gözükmektedir.

3.6. Knime ile Sınıflandırma Aşaması

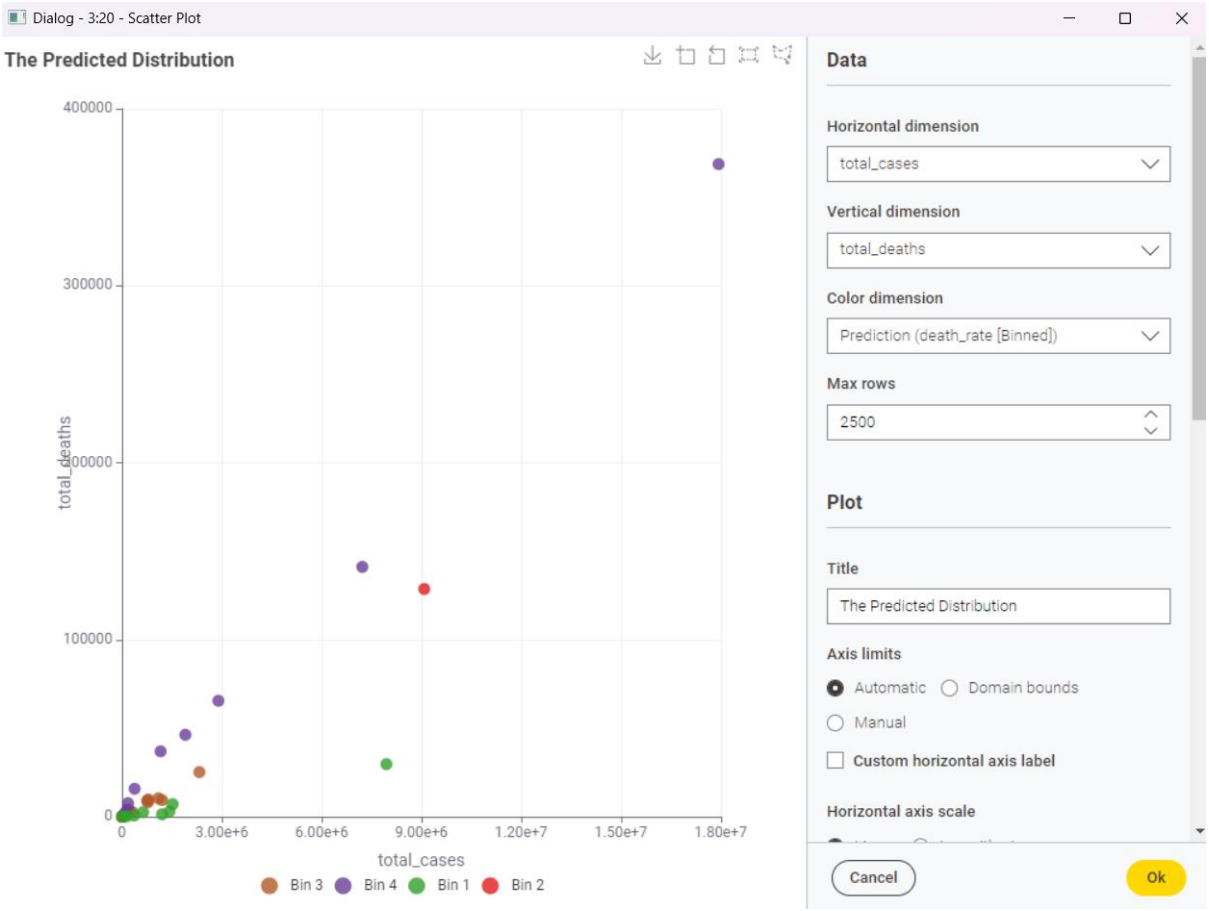


3.6.1. Şekil 31

Şekil 31'de görüldüğü üzere Knime platformu özelinde SVM modeli denenmiş ve uygulanmıştır. Şekil 32 ve Şekil 33'te de görüldüğü üzere ardından görselleştirmelere geçilmiş ve tahmin edilen dağılım ile gerçek olan dağılım karşılaştırılmıştır.


Dialog - 3:19 - Scatter Plot
The Actual Distribution
400000
300000
200000
100000
0
total_deaths
0
3.00e+6
6.00e+6
9.00e+6
1.20e+7
1.50e+7
1.80e+7
total_cases
Bin 2
Bin 4
Bin 3
Bin 1
Data
Horizontal dimension
total_cases
Vertical dimension
total_deaths
Color dimension
death_rate [Binned]
Max rows
2500
Plot
Title
The Actual Distribution
Axis limits
Automatic
Domain bounds
Manual
Custom horizontal axis label
Horizontal axis scale
Cancel
Ok

3.6.2. Şekil 32



3.6.3. Şekil 33

Jupiter arayüzünde yapılana kıyasla gerçek veri seti ile tahmin edilen arasındaki farklar daha gözlegörülebilir bir sonuç vermiştir. Buna sebep olarak default ayarların farklı olması ve düğüm opsiyonlarının doğru ayarlanması gösterilebilir.

3.7. Sonuç

Sınıflandırma aşaması ile sayısal verilere sahip bir veri seti üzerinden nasıl sınıflandırma işlemi yapılabileceği, hangi sınıflandırma modellerinin nasıl çalıştığı dolayısıyla hangi durumlarda kimin daha iyi gideceği gibi konular denenmiş ve analiz edilmiştir. Bu süreç boyunca çeşitli görsellendirmeler yardımıyla süreç daha açık bir hale getirilmiş ve anlaması daha kolay bir hale sokulmuştur. Bu süreç boyunca elde edilen grafiklerden ise veri setine dair önemli bilgiler edinilmiştir.

4. CLUSTERING

Kümeleme algoritmasında daha önceden hazırlanan *risk_rate* dataframe'i faydalanılmış ve *risk_rate* sütunu üzerinde **Şekil 34**'te de gözüktüğü üzere KMeans algoritması kullanılarak bir kümeleme işlemi gerçekleştirilmiştir.

4.1. KMeans Modelinin Fit Edilmesi

```
[142]: from sklearn.cluster import KMeans
       from sklearn.preprocessing import StandardScaler

[143]: scaler = StandardScaler()

[144]: X_cluster = risk_rate[['risk_rate']]

[145]: X_scaled = scaler.fit_transform(X_cluster)

       k_optimal = 4
       kmeans = KMeans(n_clusters=k_optimal, random_state=42, n_init=10)
       risk_rate['clustered'] = kmeans.fit_predict(X_scaled)

       C:\Users\User\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: Us
```

4.1.1. Şekil 34

KMeans algoritmamız sklearn kütüphanesinden çekilmiş ve *risk_rate* sütunumuz scale edildikten sonra eğitilmiştir. *StandardScaler()* sınıfının buradaki gerekliliği kümeleme algoritmalarının uzaklığa bağlı olarak çalışmasıdır. Daha yüksek değerlere bağlı olarak sapma yaşanmaması adına noktalar arasındaki mesafeler scale edilerek belli bir aralığa çekilir. **Şekil 34**'te görüldüğü üzere k değeri 4 verilmiş yani 4 farklı kümeye ayırması istenilmiştir. Ardından bu değerler **Şekil 35**, **Şekil 36** ve **Şekil 37**'de görüldüğü üzere görseller yardımıyla gerçek veri seti ile karşılaştırılmış ve gözlemlenmiştir.

4.2. Model Sonuçlarının Görselleştirilmesi

```
[146]: risk_rate[risk_rate['clustered'] == 0]
```

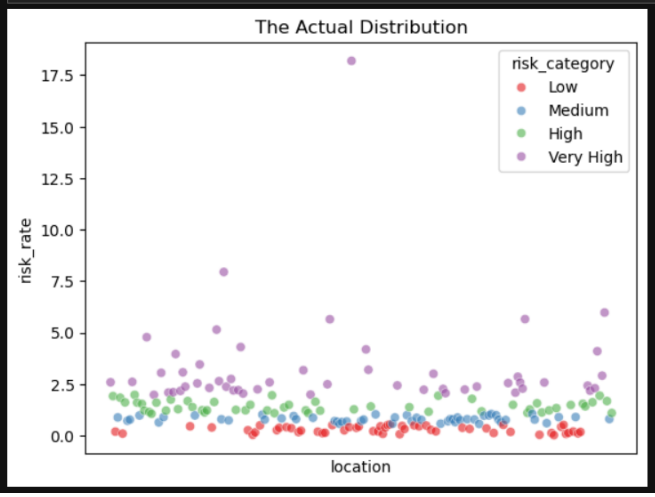
[146]:

	continent	location	total_cases	total_deaths	risk_rate	risk_category	clustered
2	Africa	Benin	86394.0	163.0	0.188671	Low	0
3	Africa	Botswana	305984.0	2688.0	0.878477	Medium	0
5	Africa	Burundi	40552.0	38.0	0.093707	Low	0
7	Africa	Cape Verde	56031.0	401.0	0.715675	Medium	0
8	Africa	Central African Republic	14649.0	113.0	0.771384	Medium	0
...	...	...	...	...	...	...	...
213	Oceania	Tonga	10494.0	11.0	0.104822	Low	0
215	Oceania	Vanuatu	7533.0	13.0	0.172574	Low	0
217	South America	Argentina	9083673.0	128653.0	1.416310	High	0
228	South America	Uruguay	899723.0	7210.0	0.801358	Medium	0
229	South America	Venezuela	522514.0	5709.0	1.092602	High	0

134 rows × 7 columns

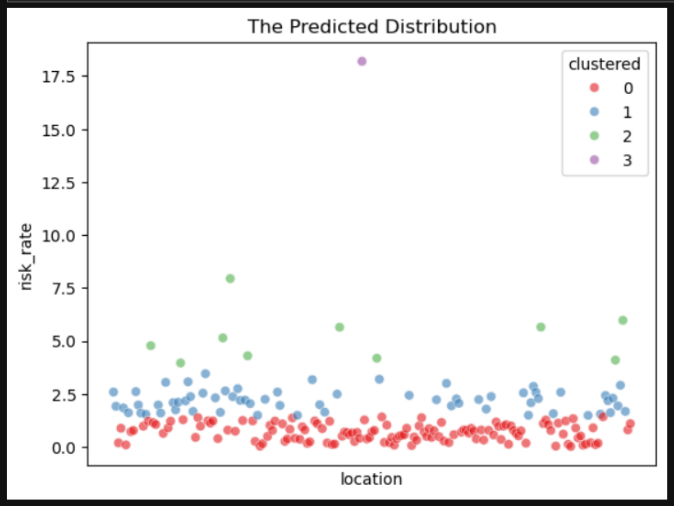
4.2.1. Şekil 35

```
[147]: sns.scatterplot(x = risk_rate.location, y = risk_rate.risk_rate, hue = risk_rate.risk_category, alpha=0.6, palette='Set1')
       plt.xticks([])
       plt.title("The Actual Distribution")
       plt.show()
```



4.2.2. Şekil 36

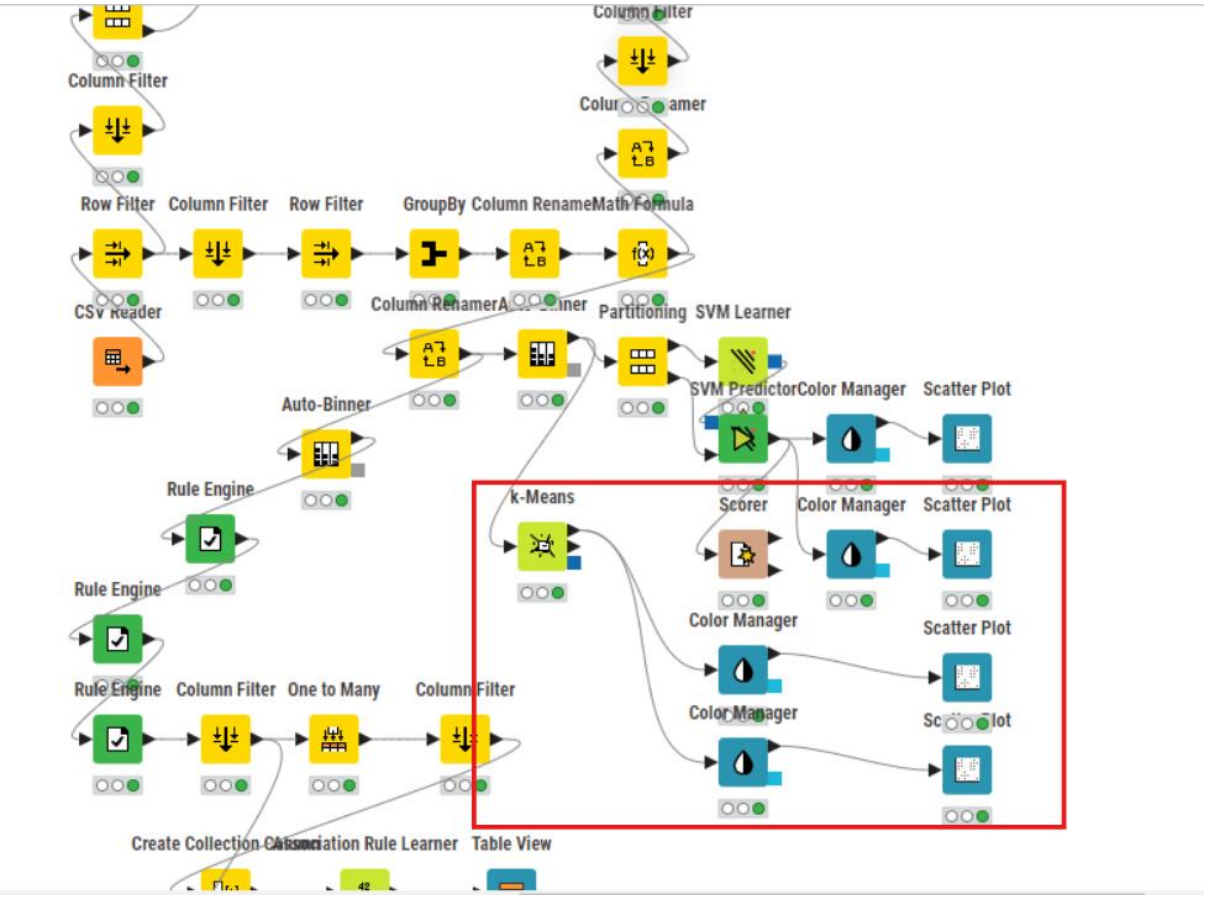
```
[148]: sns.scatterplot(x = risk_rate.location, y = risk_rate.risk_rate, hue = risk_rate.clustered, alpha=0.6, palette='Set1')
       plt.xticks([])
       plt.title("The Predicted Distribution")
       plt.show()
```



4.2.3. Şekil 37

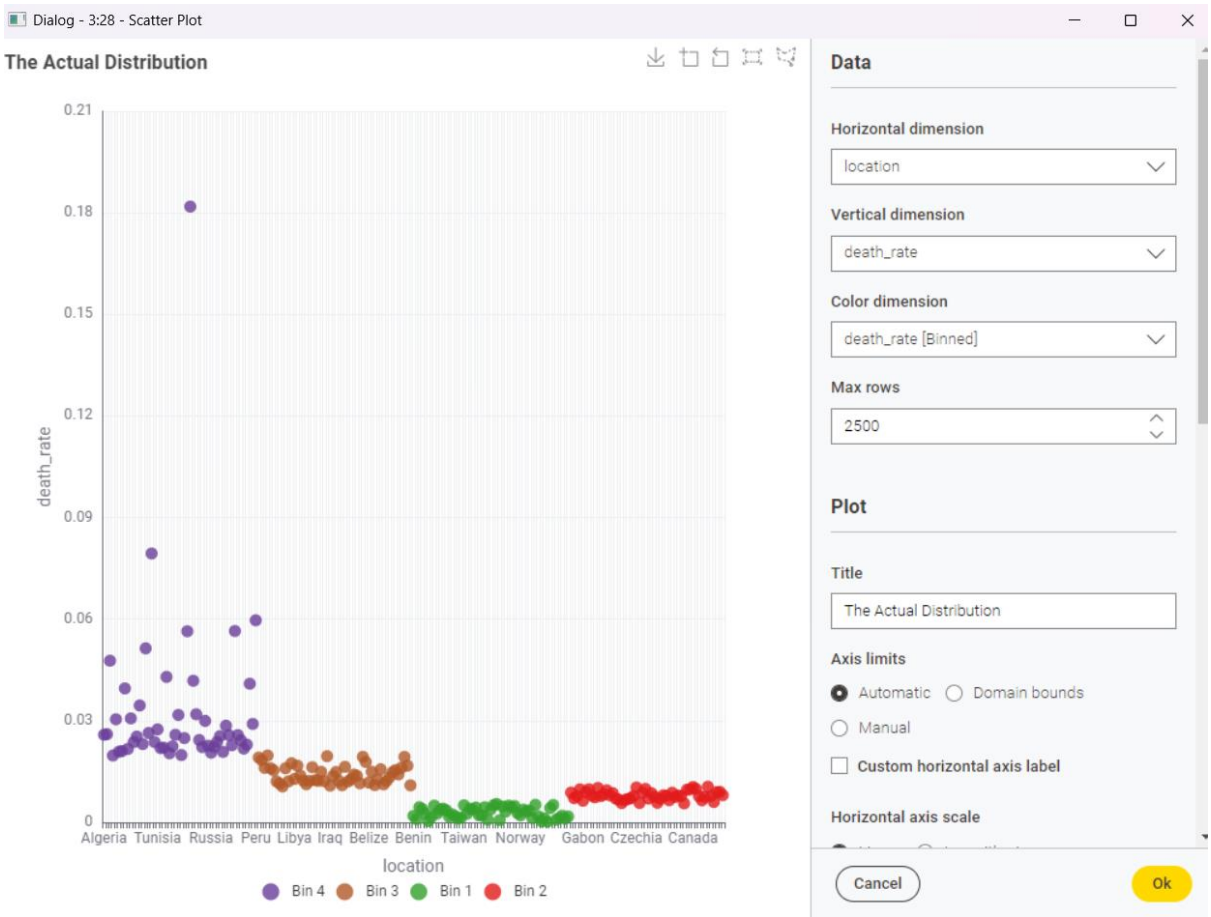
Gerçek veri seti ile tahmin edilen veri seti arasındaki farkları **Şekil 36** ve **Şekil 37** karşılaştırıldığı takdirde net bir şekilde görebilmekteyiz. Kümeleme algoritmamız sınıf ayrımlarını daha geniş bir skalada değerlendirirken gerçek veri setimiz ise düzenli aralıklara sahiptir.

4.3. Knime ile Kümeleme Problemi

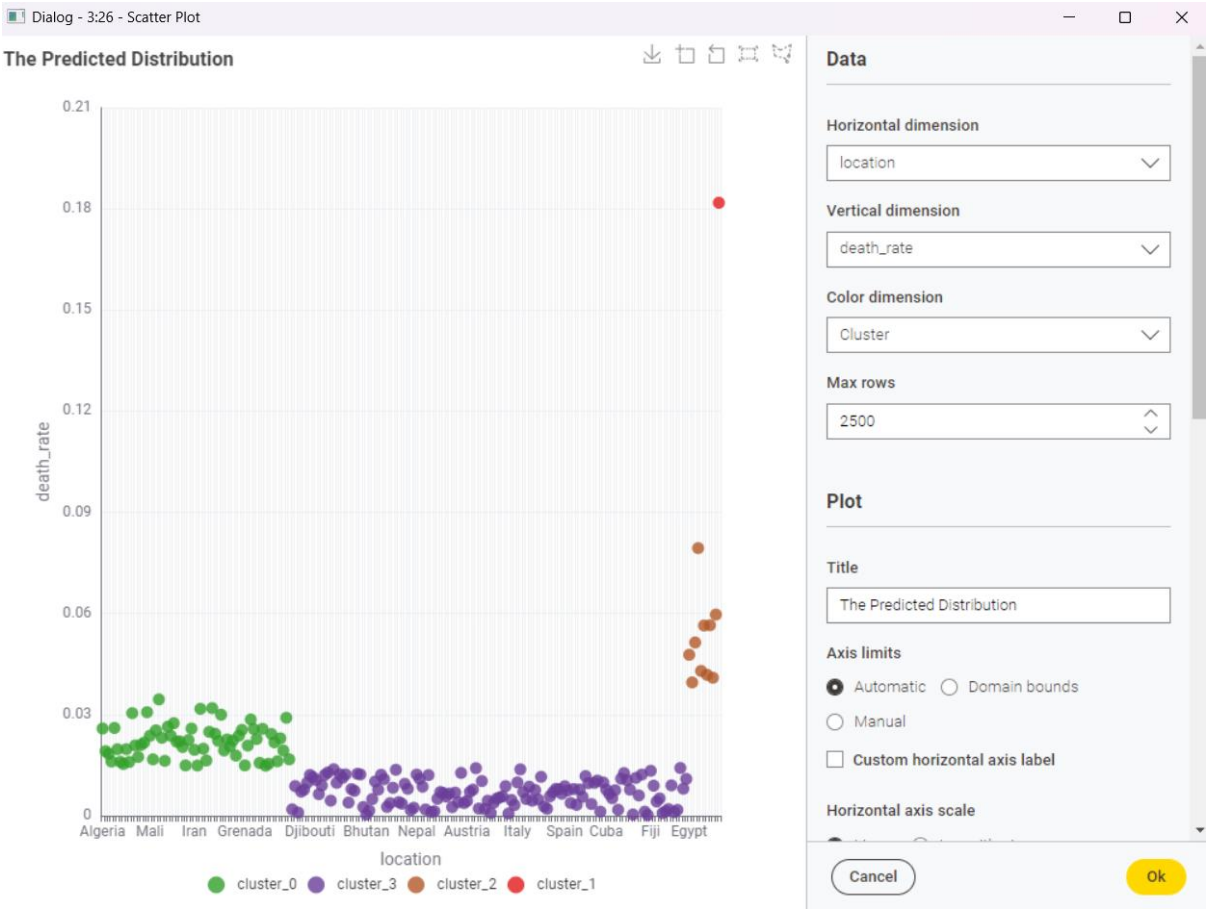


4.3.1. Şekil 38

Şekil 38'de görüldüğü üzere *k-Means* düğümü ile tahmin yapılmış ve daha sonrasında *Scatter Plot* düğümü ile gerçek veri seti ile tahmin edilen veri arasındaki farklar incelenmiştir.


Dialog - 3:28 - Scatter Plot
The Actual Distribution
0.21
0.18
0.15
0.12
0.09
0.06
0.03
0
death_rate
Algeria Tunisia Russia Peru Libya Iraq Belize Benin Taiwan Norway Gabon Czechia Canada
location
Bin 4 Bin 3 Bin 1 Bin 2
Data
Horizontal dimension
location
Vertical dimension
death_rate
Color dimension
death_rate [Binned]
Max rows
2500
Plot
Title
The Actual Distribution
Axis limits
Automatic
Domain bounds
Manual
Custom horizontal axis label
Horizontal axis scale
Cancel
Ok

4.3.2. Şekil 39



4.3.3. Şekil 40

Şekil 39 ve **Şekil 40** aracılığıyla gerçek ve tahmin edilen veri setlerin arasındaki farklara ulaşabiliriz. Kümelemeyle alakalı yine gerçek veriden bağımsız bir kümelenme örneğinin oluştuğu görülmektedir. Veri setinden izole değerlerin kümeleme problemi için kafa karıştırıcı bir etkisi olduğunu da inceleyebilmekteyiz.

4.4. Sonuç

Kümeleme algoritması ile çoktan sınıflarına ayrılan bir veri seti tekrardan kümelenmiş ve arasındaki farklar değerlendirilmiştir. Bu sayede kümeleme algoritmalarının veri setine gerçekten farklı olarak nasıl baktığı incelenmiş ve farklı bir bakış açısı kazanılmıştır.

5. Association Rule Mining

Association Rule Mining özelinde yapılan çalışmalara ilk olarak **Şekil 41**'de görüldüğü üzere daha öncesinde oluşturulmuş olan *risk_rate* dataframe'i kopyalanmış ve devamında gerekli binning işlemleri ve dummy işlemleri ile devam edilmiştir.

5.1. DataFrame oluşturulması ve Binning İşlemleri

```
[150]: associate_rule = risk_rate.copy()

[151]: associate_rule['Total_Cases'] = pd.qcut(associate_rule['total_cases'], q = 3, labels = ['Low', 'Medium', 'High'])
       associate_rule['Total_Deaths'] = pd.qcut(associate_rule['total_deaths'], q = 3, labels = ['Low', 'Medium', 'High'])
       associate_rule.head()
```

[151]:

	continent	location	total_cases	total_deaths	risk_rate	risk_category	clustered	Total_Cases	Total_Deaths
0	Africa	Algeria	265782.0	6875.0	2.586706	Very High	1	Medium	Medium
1	Africa	Angola	99287.0	1900.0	1.913644	High	1	Medium	Medium
2	Africa	Benin	86394.0	163.0	0.188671	Low	0	Medium	Low
3	Africa	Botswana	305984.0	2688.0	0.878477	Medium	0	Medium	Medium
4	Africa	Burkina Faso	20865.0	383.0	1.835610	High	1	Low	Low

5.1.1. Şekil 41

Şekil 41'de görüldüğü üzere *total_cases* ve *total_deaths* sütunlarına da binning işlemleri çeyrekler özelinde uygulanmış, ardından *Total_Cases* ve *Total_Deaths* olarak kaydedilmiştir. Ardından get dummies methodu ile **Şekil 42**'de görüldüğü üzere *Total_Cases*, *Total_Deaths* ve *risk_category* sütunları için dummylere ayırma işlemi gerçekleştirilmiştir.

5.2. Sütunlar Özelinde Kategorileri Kodlama

```
[152]: df_encoded = pd.get_dummies(associate_rule[['Total_Cases', 'Total_Deaths', 'risk_category']])

[153]: df_encoded.head()
```

[153]:

	Total_Cases_Low	Total_Cases_Medium	Total_Cases_High	Total_Deaths_Low	Total_Deaths_Medium	Total_Deaths_High	risk_category_Low	risk_category_Medium	risk_
0	False	True	False	False	True	False	False	False	
1	False	True	False	False	True	False	False	False	
2	False	True	False	True	False	False	True	False	
3	False	True	False	False	True	False	False	True	
4	True	False	False	True	False	False	False	False	

5.2.1. Şekil 42

Şekil 42'de gördüğümüz üzere gerekli sütunlar özelinde kategorik kodlama işlemi gerçekleşmiş ve sütunların özelindeki sınıflar farklı sütunlar konumuna oturmuştur. Daha sonrasında elde edilen boolean değerler **Şekil 43**'te de gözüktüğü üzere daha net bir görünüm elde etmek adına integer değerlere dönüştürülmüştür.

```
[154]: df_encoded = df_encoded.astype(int)

[155]: df_encoded.head()
```

[155]:

	Total_Cases_Low	Total_Cases_Medium	Total_Cases_High	Total_Deaths_Low	Total_Deaths_Medium	Total_Deaths_High	risk_category_Low	risk_category_Medium	risk_
0	0	1	0	0	1	0	0	0	
1	0	1	0	0	1	0	0	0	
2	0	1	0	1	0	0	1	0	
3	0	1	0	0	1	0	0	1	
4	1	0	0	1	0	0	0	0	

5.2.2. Şekil 43

Görüldüğü üzere *False* ve *True* gibi değerler gitmiş ve onların yerine *0* ve *1* gibi integer değerler gelmiştir. Ardından veri setimiz **Şekil 44**'te görüldüğü üzere Association Rule modeline fit edilmiş ve çalıştırılmıştır.

5.3. Modelin Fit Edilmesi

```
[156]: from mlxtend.frequent_patterns import fpgrowth
       from mlxtend.frequent_patterns import apriori, association_rules

       frequent_itemsets = fpgrowth(df_encoded, min_support=0.1, use_colnames=True)
       rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)

       C:\Users\User\anaconda3\Lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarn
```

5.3.1. Şekil 44

Şekil 44'te görüldüğü üzere Association Rule modelleri önce mlxtend kütüphanesinden çekilmiş, daha sonrasında fit edilmiş ve *frequent_itemsets* ile *rules* nesneleri oluşturulmuştur. Ardından **Şekil 45** ve **Şekil 46**'da görüldüğü üzere bu nesnelerin çıktılarına bakılmış ve arasındaki bağlar incelenmiştir.

5.4. Modelin Sonuçlarının İncelenmesi

```
[157]: frequent_itemsets
```

[157]:

	support	itemsets
0	0.330144	(Total_Deaths_Medium)
1	0.330144	(Total_Cases_Medium)
2	0.248804	(risk_category_Very High)
3	0.248804	(risk_category_High)
4	0.334928	(Total_Deaths_Low)
5	0.253589	(risk_category_Low)
6	0.248804	(risk_category_Medium)
7	0.334928	(Total_Cases_Low)
8	0.334928	(Total_Deaths_High)
9	0.334928	(Total_Cases_High)
10	0.229665	(Total_Deaths_Medium, Total_Cases_Medium)
11	0.110048	(risk_category_Very High, Total_Cases_Medium)
12	0.119617	(risk_category_Very High, Total_Deaths_High)
13	0.124402	(Total_Deaths_Low, risk_category_Low)
14	0.100478	(Total_Cases_Low, risk_category_Low)
15	0.100478	(Total_Cases_Low, Total_Deaths_Low, risk_categ...
16	0.110048	(Total_Cases_High, risk_category_Medium)
17	0.100478	(risk_category_Medium, Total_Deaths_High)
18	0.296651	(Total_Cases_Low, Total_Deaths_Low)
19	0.272727	(Total_Cases_High, Total_Deaths_High)

5.4.1. Şekil 45

```
[158]: rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
```

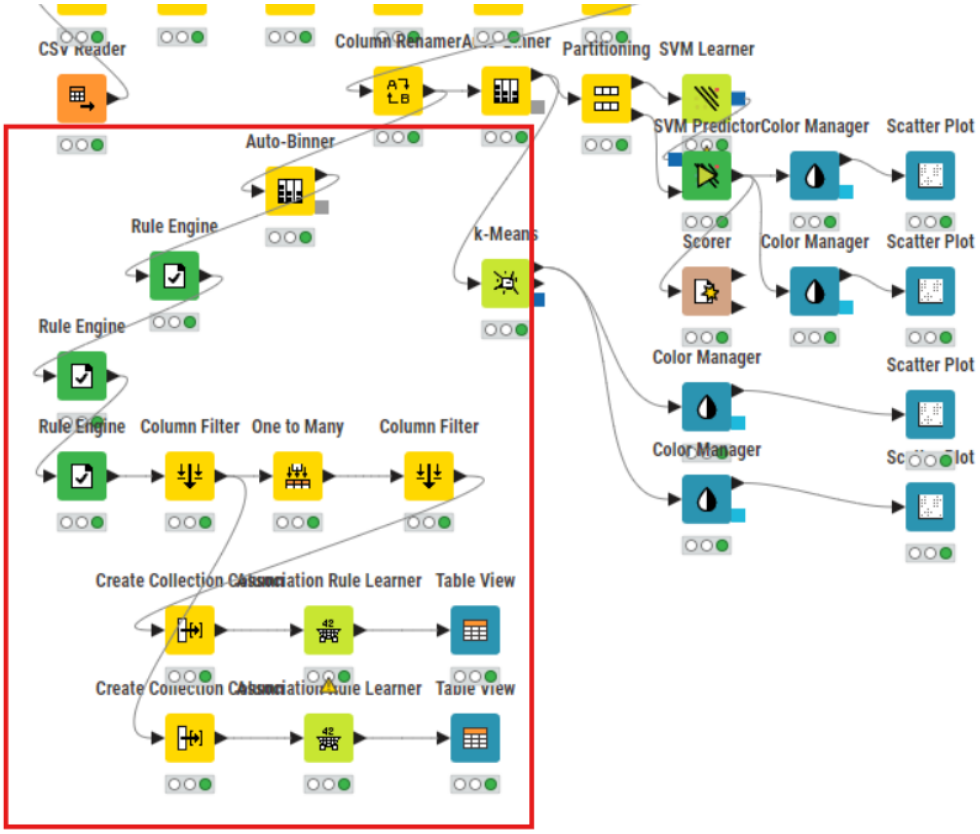
[158]:

	antecedents	consequents	support	confidence	lift
0	(Total_Deaths_Medium)	(Total_Cases_Medium)	0.229665	0.695652	2.107120
1	(Total_Cases_Medium)	(Total_Deaths_Medium)	0.229665	0.695652	2.107120
2	(Total_Cases_Low, risk_category_Low)	(Total_Deaths_Low)	0.100478	1.000000	2.985714
3	(Total_Deaths_Low, risk_category_Low)	(Total_Cases_Low)	0.100478	0.807692	2.411538
4	(Total_Cases_Low)	(Total_Deaths_Low)	0.296651	0.885714	2.644490
5	(Total_Deaths_Low)	(Total_Cases_Low)	0.296651	0.885714	2.644490
6	(Total_Cases_High)	(Total_Deaths_High)	0.272727	0.814286	2.431224
7	(Total_Deaths_High)	(Total_Cases_High)	0.272727	0.814286	2.431224

5.4.2. Şekil 46

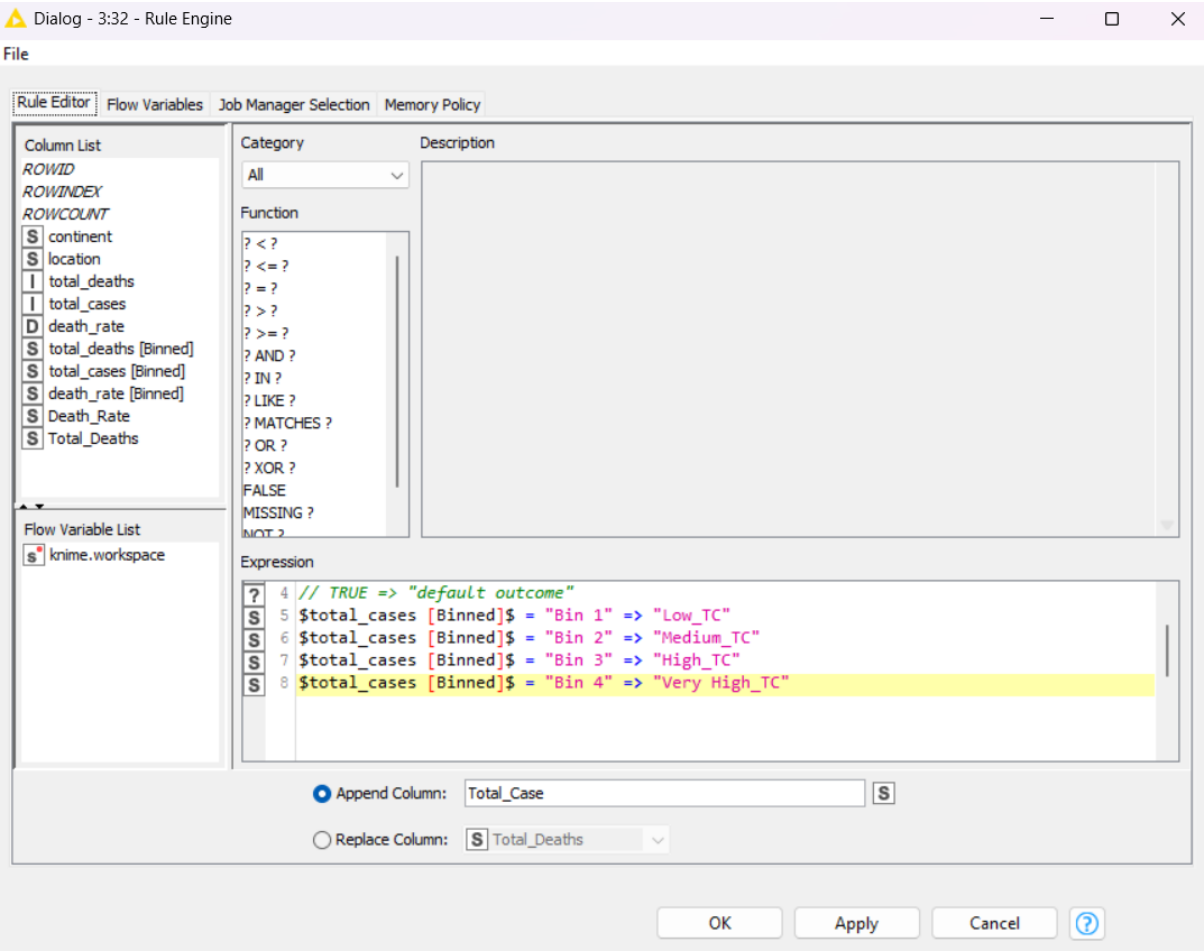
Veri setinden elde edilen ilginç bilgilere ise **Şekil 45** ve **Şekil 46** aracılığıyla ulaşabiliyoruz. Buradan görüldüğü üzere *Total_Deaths* ve *Total_Cases* değişkenlerinin yüksek olması *risk_category* değişkenimizin düşük olma ihtimalini oldukça düşüren bir yapıya sahip. Bu mantığı *2÷3* sayısının *3÷4* sayısından daha küçük olması ile daha iyi anlayabiliriz. *Total_Deaths* ve *Total_Cases* değerleri sürekli olarak lineer bir ilişki çerçevesinde arttıkları için oranları da artmaktadır. Bu da *risk_category* nesnemizin düşük çıkma ihtimalini oldukça düşürmektedir.

5.5. Knime ile Association Rule Mining



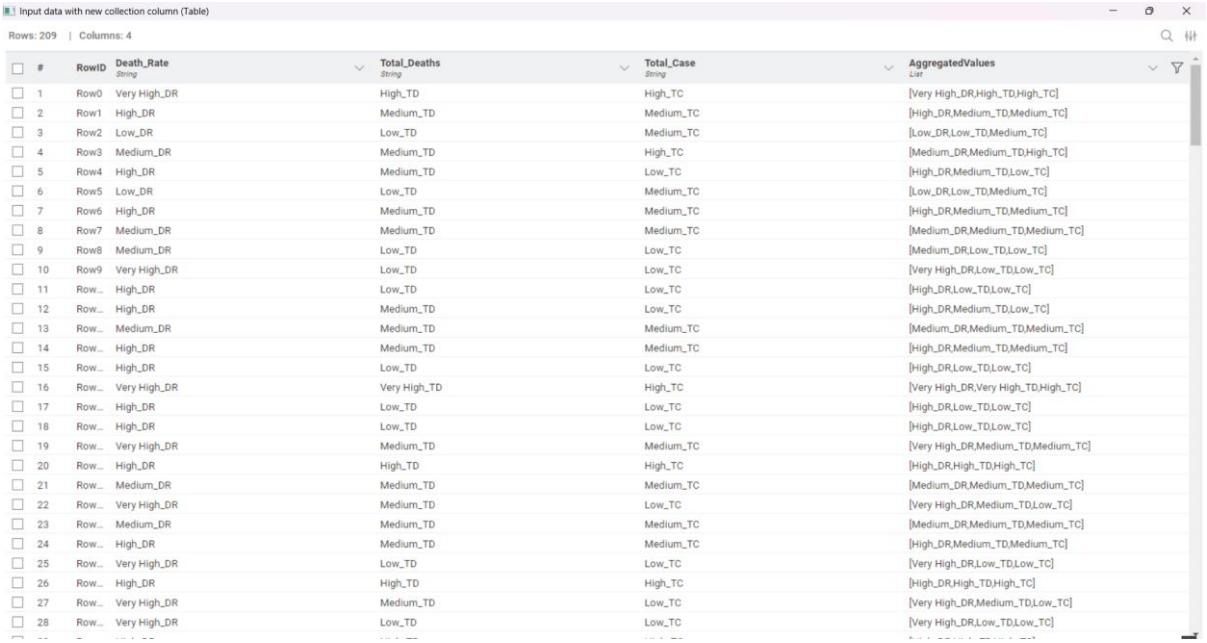
5.5.1. Şekil 47

Şekil 47'de görüldüğü üzere öncesinde binning işlemi uygulanan sütunları rule engin ile **Şekil 48**'de görünecek şekilde isim değişimleri uygulanmıştır.



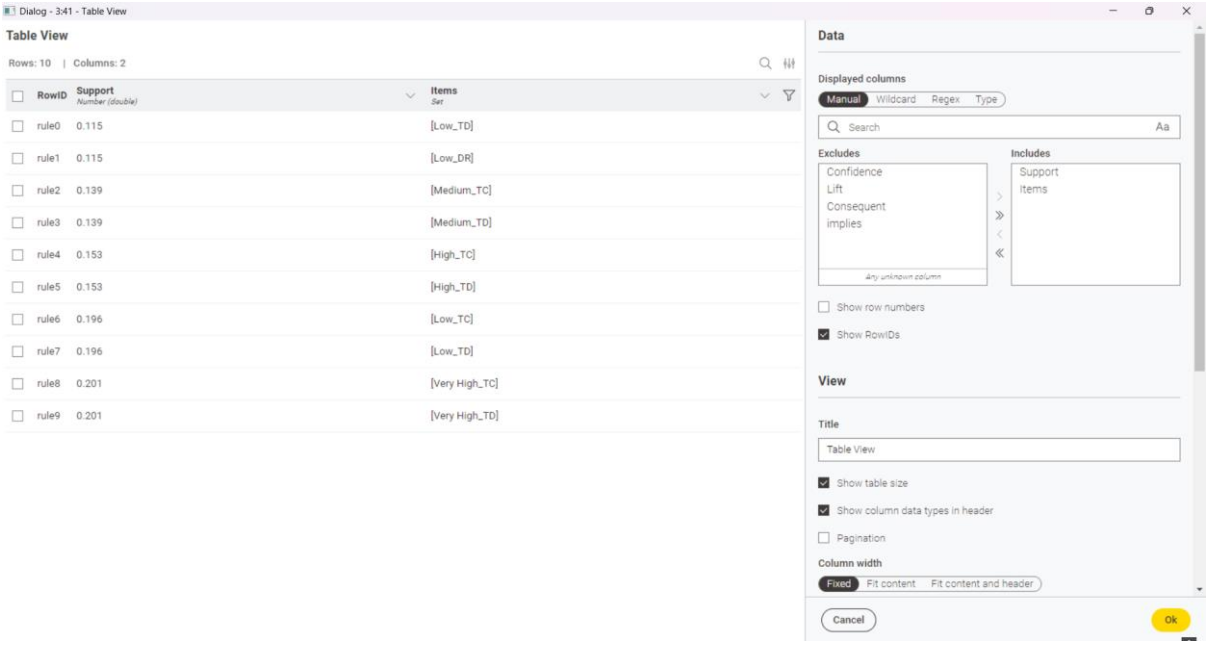
5.5.2. Şekil 48

Ardından **Şekil 49**'da görülebileceği üzere *Create Collection Column* ile sütunları satır bazlı ek bir dizi olacak şekilde yeni bir sütun oluşturulmuştur.

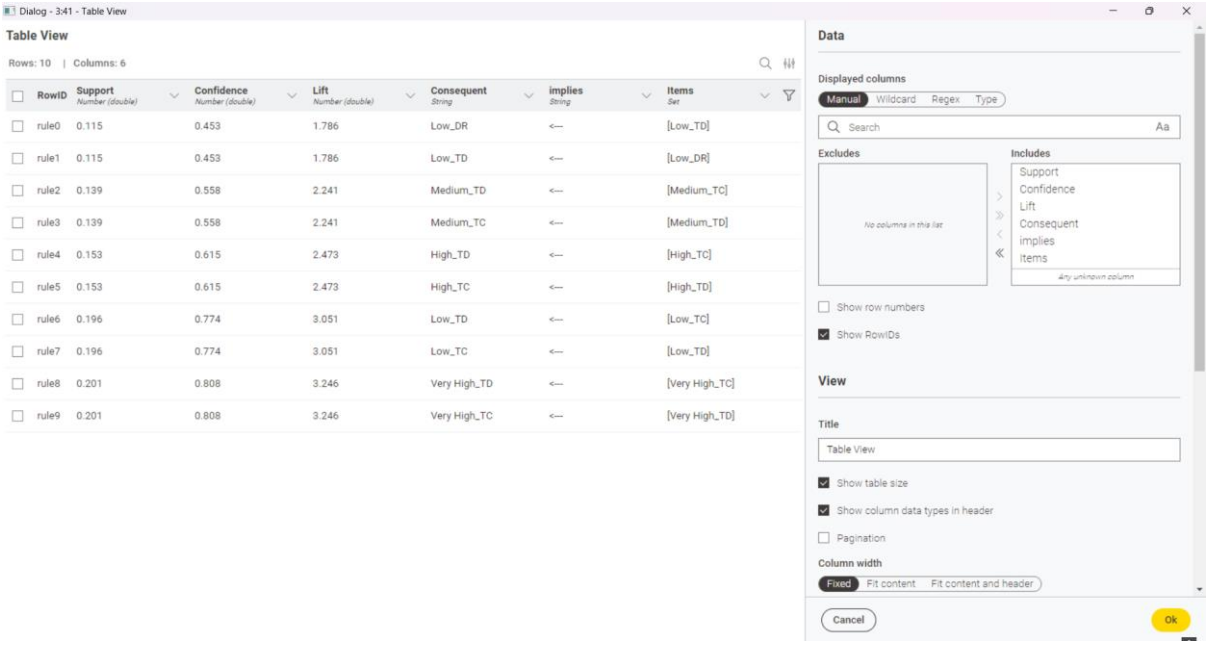


5.5.3. Şekil 49

Daha sonrasında bu sütun yani **Şekil 49**'daki *AggregatedValues* sütunu *Association Rule Learner* düğümüne verilmiş ve çıktısına **Şekil 50** ve **Şekil 51**'de görebileceğimiz üzere *Table View* düğümüyle bakılmıştır.



5.5.4. Şekil 50



5.5.5. Şekil 51

Görüldüğü üzere Support, Confidence ve Lift değerlerimize yine ulaşmış olduk.

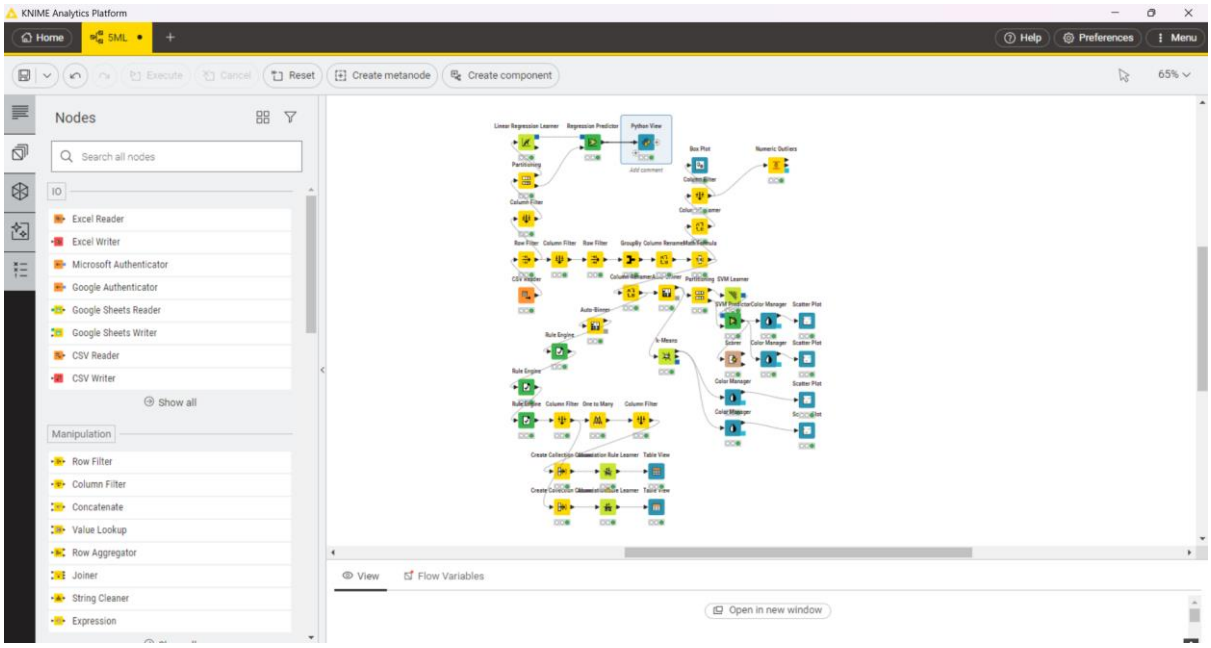
5.6. Sonuç

Birliktelik Kural Çıkarımı yöntemi ile veri setinde gözükmeyen bilgileri toplamış ve ilişkileri istatistiksel bazda değerlendirmiş olduk. Veri seti üzerinde daha geniş bir bilgi yelpazesine sahip olmamızı ve daha rahat çıkarımlar yapmamızı sağlayan bu yöntem aynı zamanda takip ettiği adımlar açısından da öğretici olmuştur.

SONUÇ

Çeşitli Makine Öğrenimi ve Veri Analizi adımlarını Covid-19 temelli sağlık sektörüne yönelik bir veri setiyle uygulanmıştır. Bu süreç boyunca görselleştirmeler, dönüşümler, istatistiki formüller, sektörel bilgiler ve keşifsel veri analizleri uygulanmıştır. Her bir başlıktan farklı kazanımlar elde edilmiş ve veri setine dair ilginç bilgileri farklı bakış açılarıyla görme fırsatı yakalanılmıştır.

Knime Over View



6. EK BAŞLIKLAR

Auto ML ve Hiperparameter Tuning bölümleri dokümantasyonda yer almamıştır. Bunların sebepleri ise aşağıdaki başlıklarda verilmiştir.

6.1. Auto ML

Auto ML özelinde bir alt kütüphanesi olan TPOT kütüphanesi denenmiş ve onun özelinde lineer problemler için lineer sınıfı ve classificationlar içinse başka bir sınıfı denenmiştir ve **Şekil 52** ve **Şekil 53**'te görüldüğü üzere gerçekten kötü sonuçlar verildiği gözlemlenmiştir. Veri setinin görece büyük olmasından kaynaklı da her bir eğitim aşaması bir hayli yorucu geçmiş ve optimization ile düzelecek bir hata düzeltme sürecine ise benzememeye başlanmıştır. Bundan ötürü daha sonrasında manuel olarak bir lineer regression fit edilmiş ve o şekilde ilerlenme kararı alınmıştır.

```
[41]: X = world_data[['total_cases', 'total_deaths', 'death_momentum', 'death_rate']]
      y = world_data['new_deaths']

[42]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

      tpot = TPOTRegressor(generations=5, population_size=20, verbosity=2, random_state=42)

      tpot.fit(X_train, y_train)

      print("R² Score:", tpot.score(X_test, y_test))

      tpot.export('best_pipeline.py')

      Version 0.12.2 of tpot is outdated. Version 1.0.0 was released Wednesday February 26, 2025.
      Loading widget...

      Generation 1 - Current best internal CV score: -1895532.8104068576

      Generation 2 - Current best internal CV score: -1895532.8104068576

      Generation 3 - Current best internal CV score: -1795558.220174618

      Generation 4 - Current best internal CV score: -1795558.220174618

      Generation 5 - Current best internal CV score: -1795558.220174618

      Best pipeline: ExtraTreesRegressor(XGBRegressor(input_matrix, learning_rate=0.1, max_depth=4, min_child_weight=2, n_estimators=100, n_jobs=1, objective
      =reg:squarederror, subsample=0.8, verbosity=0), bootstrap=False, max_features=0.8500000000000001, min_samples_leaf=12, min_samples_split=7, n_estimator
      s=100)
      R² Score: -1505918.848052683
```

6.1.1. Şekil 52

```
[62]: X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.4, random_state=40)

[63]: tpot2 = TPOTRegressor(generations=5, population_size=20, verbosity=2, random_state=40)
      tpot2.fit(X1_train, y1_train)

      Version 0.12.2 of tpot is outdated. Version 1.0.0 was released Wednesday February 26, 2025.
      Loading widget...

      Generation 1 - Current best internal CV score: -462377893.8284175

      Generation 2 - Current best internal CV score: -306513293.7874289

      Generation 3 - Current best internal CV score: -306513293.7874289

      Generation 4 - Current best internal CV score: -306513293.7874289

      Generation 5 - Current best internal CV score: -267896776.4431901

      Best pipeline: AdaBoostRegressor(RidgeCV(XGBRegressor(input_matrix, learning_rate=0.5, max_depth=4, min_child_weight=14, n_estimators=100, n_jobs=1, ob
      jective=reg:squarederror, subsample=0.55, verbosity=0)), learning_rate=1.0, loss=square, n_estimators=100)
[63]: TPOTRegressor
      TPOTRegressor(generations=5, population_size=20, random_state=40, verbosity=2)
```

6.1.2. Şekil 53

6.2. Hiperparameter Tuning

Hiperparameter tuning özelinde ise ele alınan veri seti büyüklük ve detay açısından gayet yeterli olsa da feature türleri yeteri kadar zengin olmamasından kaynaklı yani çoğunlukla numerik veri içermesinden ve bir zaman indeksine bağlı olmasından dolayı bazı problemlerin bazı modellerin algoritmasına ters düştüğü hatta classification kısmında da görüldüğü üzere hiçbirinin SVC algoritmasına yaklaşılmadığı fark edilmiştir. Bundan ötürü hiperparameter tuning gibi hata oranını optimize etme kısımlarına düşük doğruluk veren algoritmaların neden düşük doğruluk verdiği **Şekil 24**'te de görüldüğü üzere bilindiğinden dolayı girilmemiştir.

6.3. Sonuç

Auto ML ve Hiperparameter Tuning kullanımları eldeki veri seti türü ve akışından kaynaklı istenen problemlerde kullanılmamış ve bu alanlara girilmesinin Knime tarafında da daha yorucu ve çıkılamaz bir sürece tetikleyebileceği düşünülmüştür.