

به نام خدا

لینک کولب : [data mining project phase2](#)

بخش مدل های پیش بینی

مدل **Linear Regression** (پیش بینی کرایه کل (Total Fare) براساس وزن ، مسافت و نوع محصول)

مثل مدل رگرسیون که در نوت بوک کولب نوشته بودیم ، در ابتدا کتابخانه های مورد نیاز را ایمپورت می کنیم. سپس ستون های وزن ، مسافت و نوع محصول را در یک دیتا فریم می ریزیم به صورت خط رو به رو : `selected_columns = transport_df[['Weight' , 'Distance' , 'Product code']]`. سپس در X یعنی مقادیر مستقل قرار می دهیم تا با استفاده این بتوانیم ستون وابسته را پیش بینی کنیم . ۷ را ستون کرایه کل قرار می دهیم زیرا می خواهیم کرایه کل را پیش بینی کنیم. در ادامه داده ها را به دو دسته آموزش و تست تقسیم بندی می کنیم و فقط ۳۰٪ از داده ها را به تست تخصیص می دهیم. سپس مدل را طبق کدهای موجود در لینک کولب می سازیم و مدل را آموزش می دهیم. بعد از آموزش باید مدل را به کمک داده هایی که برای بخش تست قرار دادیم ، پیش بینی کنیم. مقادیر بدست R2 Score و Mean Squared Error (MSE) و Mean Absolute Error (MAE) را بدست می آوریم.

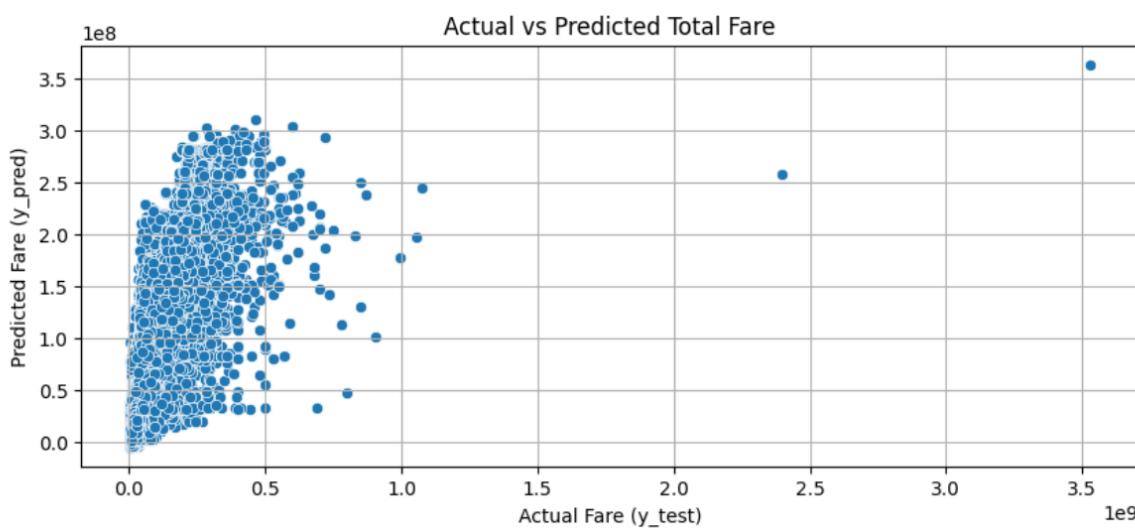
عملیات	مقدار بدست آمده
Mean Absolute Error	24632941.756846394
Mean Squared Error	1420292186426666.8
R2 Score	0.7024540167770885

MAE = 24,632,941 : یعنی به طور میانگین مدل حدود ۲۴ میلیون تومان (یا هر واحد پولی) اشتباه دارد.

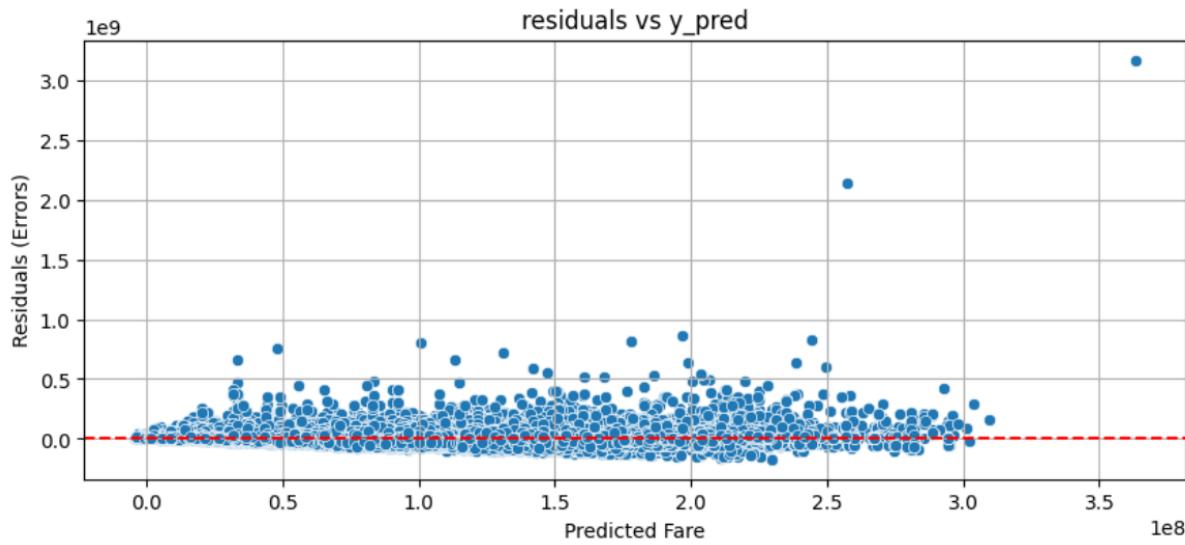
MSE = $1.42 * 10^{15}$: چون مربعی هست، خطاهای بزرگ تأثیر بیشتری دارند.
 $R^2 = 0.70$: حدود ۷۰٪ از واریانس متغیر هدف (Total Fare) توسط مدل توضیح داده شده است.
نه خیلی بد ولی می‌تواند بهتر شود.

نمودارها

نمودار Actual vs Predicted Total Fare : بیشتر نقاط در سمت چپ تجمع دارند(فاصله کمتر بین مقادیر واقعی و پیش‌بینی شده). اما در بخش‌های با کرايه بالا (Total Fare بالا) مدل دقت خوبی ندارد و پراکندگی زیاد دیده می‌شود. در نتیجه مدل برای کرايه‌های پایین عملکرد خوبی دارد ولی برای کرايه‌های خیلی بالا به خوبی آموزش داده نشده است.



نمودار Residuals vs y_pred : بیشتر خطاهای در اطراف صفر هستند، که نشان می‌دهد مدل سیستماتیک ندارد. اما باز هم چند مقدار outlier بالا هستند. مدل در کل خوب پیش‌بینی می‌کند ولی در مقدارهای بزرگ‌تر خطای زیادی دارد. ممکن‌های داده‌های پرت یا غیرنرمال در کرايه‌ها وجود داشته باشند.

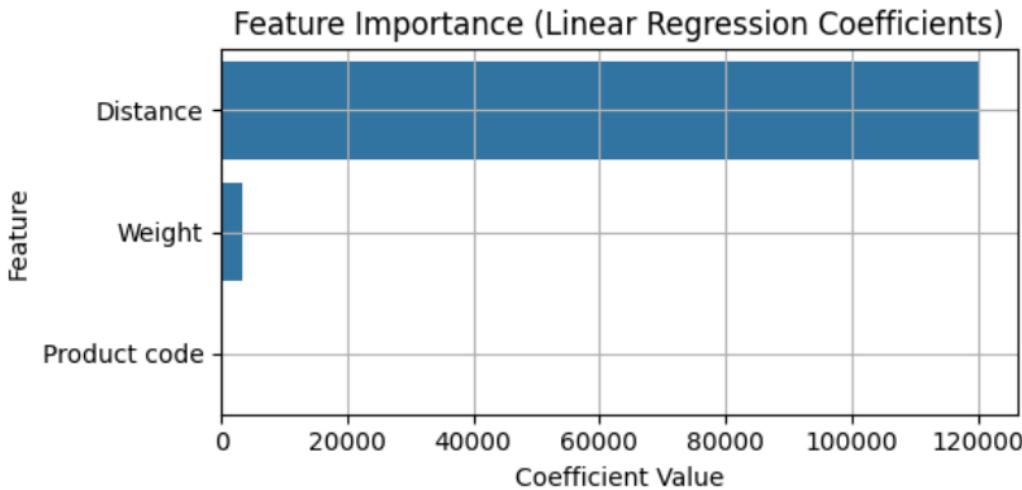


نمودار Distribution of Prediction Errors (Residuals) : اکثر خطاهای نزدیک به صفر هستند و این نشانه خوبی است. اما بعضی از خطاهای خیلی بزرگتر از بقیه هستند که در سمت راست نمودار کشیده شده اند. به این حالت می‌گیم که توزیع خطاهای Right-Skewed یا دارای دم راست هست. در نتیجه ، دارای دم راست در نمودار نشان دهنده‌ی این است که در داده‌ها دارای داده‌های پرت هستیم یا اینکه مدل به پیچیدگی بیشتری نیاز دارد.



نمودار اهمیت ویژگی‌ها `importance = model.coef_` : (Feature Importance)

ضرایب مدل را در یک متغیر به نام `importance` ذخیره می‌کنیم تا با استفاده از این متغیر بتوانیم یک دیتا فریم مرتب شده براساس اهمیت آنها ایجاد کنیم و سپس نمودار این دیتا فریم را هم رسم کنیم.



feature	Coefficient	تفسیر
Distance	خیلی بالا	قوی ترین ویژگی تاثیرگذار بر روی کرایه
Weight	دومین فاکتور مهم	وزن بر روی کرایه تاثیرگذار است و یک امر منطقی است.
Product Code	تقریباً بی تاثیر	چون مدل این مقادیر این ستون را عددی در نظر گرفته (نه معنایی) و واقعاً کد کالا تاثیری ندارد.

مدل Linear Regression برای تخمین مبلغ بیمه (Insurance Amount)

برای تخمین مبلغ بیمه از برخی از فیچرها و ستونها عددی برای متغیر مستقل استفاده کردیم که برخی از فسچرهایی که استفاده کردیم مقادیر عددی شان نشان دهنده یک کد بود که باید با روشی آنها را به مقادیر عددی معناداری تبدیل می کردیم زیرا گرچه این ستون عددی به نظر می رسد ولی در واقع نمایانگر دسته بندی و یا کد هستند پس از روش Target Encoding برای این حل این مشکل استفاده می کنیم . حال بباید کمی بیشتر این موضوع را توضیح دهیم تا به درک خوبی از این موضوع برسیم ؛ اول ستونهایی که برای متغیر مستقل استفاده کردیم را لیست می کنیم.

```

انتخاب ستون ها برای متغیر مستقل#
selected_features = [
    'Weight',
    'Distance',
    'Value added insurance premium',
    'Evacuation fee',
    'Loading fee',
    'Complications of moving goods',
    'Company commission',
    'product_code_encoded',           #instead of product code
    'packaging_code_encoded',         #instead of packaging code
    'originCity_code_encoded',        #instead of Code of the city of origin
    'destinationCity_code_encoded'   #instead of Destination city code
]

```

همانطور که مشاهده می شود در تصویر بالا لیست فیچرهایی که استفاده شده است آورده شده. اما Product code , Packaging code , Code of city of origin , Destination چون ویژگی های city code مقادیرشان عددی اما از نوع کد هستند یعنی کدهایی مانند 123 product code = 87 را فقط برچسب (label) هستند و هیچ رابطه ریاضی بین این اعداد وجود ندارد. مثلا :

آیا اینکه کد محصول ۲۰۰ بزرگ تر از ۱۰۰ هست، معنی دار هست؟ نه!
در صورتی که مدل رگرسیون عدد ۲۰۰ را واقعاً بزرگ تر و تأثیرگذارتر می داند، که اشتباه است!

: Target Encoding

در این روش به جای استفاده مستقیم از خود کد، مقدار میانگین ستون هدف یا تارگت (مثلاً (insurance amount) برای هر کد محاسبه می شود.

سپس این مقدار میانگین به جای کد اصلی در مدل مورد استفاده قرار می گیرد.

مزیت هایی که استفاده از روش Target Encoding دارد به شرح زیر است :

- حذف معنای نادرست عددی : دیگر مدل فکر نمی کند که "کد ۲۰۰" بهتر از "کد ۱۰۰" هست.

- ارتباط مستقیم با هدف : مقدار جایگزین شده دقیقاً بر اساس ارتباط با insurance amount هست.
- قابل استفاده در مدل های ساده مثل رگرسیون : چون فقط یک ستون عددی جدید به مدل اضافه می شود و به overfitting دچار نمی شویم.

حال به این می پردازیم که چگونه روش Target Encoding را بر روی ستون ها پیاده سازی کنیم. در اینجا ما فقط یک نمونه را بررسی می کنیم زیرا بقیه ای ستون ها هم همین روند را دارند. برای ستون Product code را توضیح خواهیم داد.

```
▶ مرحله #1
محاسبه میانگین مبلغ بیمه
برای هر کد محصول
product_code_target_mean = transport_df.groupby('Product code')['Insurance amount'].mean()

▶ مرحله #2
جایگزین کردن هر مقدار product code میانگین مربوط به خودش
transport_df['product_code_encoded'] = transport_df['Product code'].map(product_code_target_mean)
سپس در مدل رگرسیون در سلول بعدی از این ستون به جای ستون کد محصول به عنوان یکی از ستون های متغیر مستقل استفاده می کنیم

transport_df[['Product code' , 'product_code_encoded']]
```

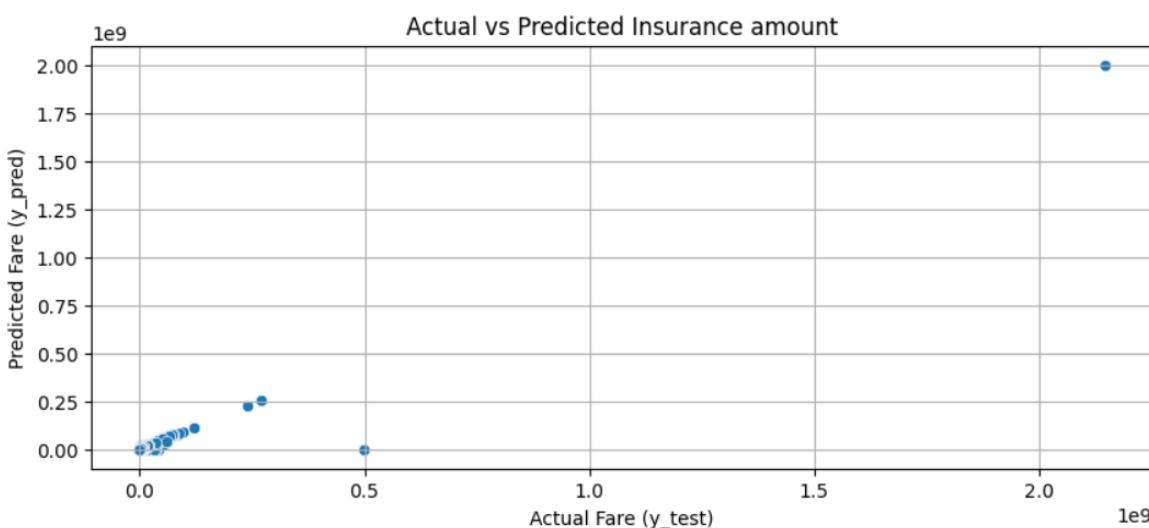
همانطور که در قطعه کد بالا در تصویر مشاهده می کنید با استفاده از groupby بر روی ستون کد محصول می آییم و میانگین مبلغ بیمه را محاسبه می کنیم و یک ستون ایجاد می کنیم به نام product_code_target_mean که در این ستون مقادیری که در product_code_encoded بدست آمد ، براساس کد محصولات روی هر سطر قرار می گیرد. و با این کار ما می توانیم با خیال راحت مدل را بسازیم .

برای ۳ ستون دیگر هم همین کار را انجام می دهیم.

مثل همیشه متغیر مستقل و وابسته را مشخص می کنیم و مدل را می سازیم و آموزش می دهیم؛ سپس مدل را پیش بینی می کنیم با داده هایی که برای بخش تست نگه داشته بودیم. سپس مقادیر MAE و MSE و R2 Score را محاسبه می کنیم تا متوجه شویم مدلی که ساختیم چقدر خوب است :

معیار	مقدار بدست آمده	تحلیل
Mean Absolute Error	117660.76742609269	میانگین خطا نسبتاً کم است. مدل به طور میانگین فقط حدود ۱۱۸ هزار خطا دارد. خوب است.
Mean Squared Error	1792863055296.0906	عدد بزرگی است چون مربع خطاهای بزرگ (outliers) تأثیر زیادی می‌گذارد. وجود برخی خطاهای بزرگ تأیید می‌شود.
R2 Score	0.9451622298732499	یعنی مدل حدود ۹۴.۵٪ از واریانس داده‌ها را توضیح می‌دهد. این مقدار نشان‌دهنده دقیق بسیار بالای مدل است.

نمودار Actual vs Predicted Insurance amount



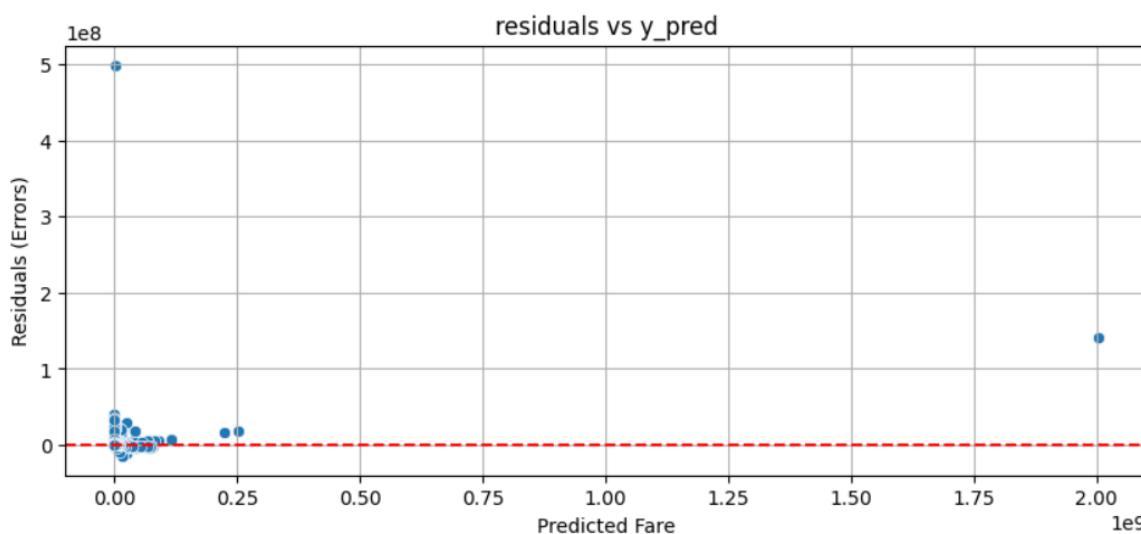
در نمودار بالا همانطور که مشاهده می شود در روی محور x مقادیر واقعی مبلغ بیمه قرار دارند. بر روی محور y مقادیر پیش بینی شده مبلغ بیمه توسط مدل قرار دارند.

تحلیل : بیشتر نقاط پایین سمت چپ نمودار تجمع دارند و تقریباً روی خط $y = x$ قرار دارند که نشان می دهد مدل اکثر پیش بینی ها را درست انجام داده است. اما یک نقطه پرت (Outliers) در

بالا سمت راست نمودار دیده می شود که مقدار واقعی آن بسیار بالاست اما مقدار پیش بینی شده اش خیلی پایین تر از آن است. همین طور یک نقطه هم در قسمت پایین و سمت چپ نمودار دیده می شود که دور از سایر نقاط افتاده و مقدار واقعی آن حدودا ۵۰ میلیون بوده اما مقدار پیش بینی شده اش + است که خیلی تفاوت دارد.

در کل این مدل خوب عمل کرده است اما برای مقادیر بسیار بالا ضعف داشته است.

نمودار residual vs y_pred



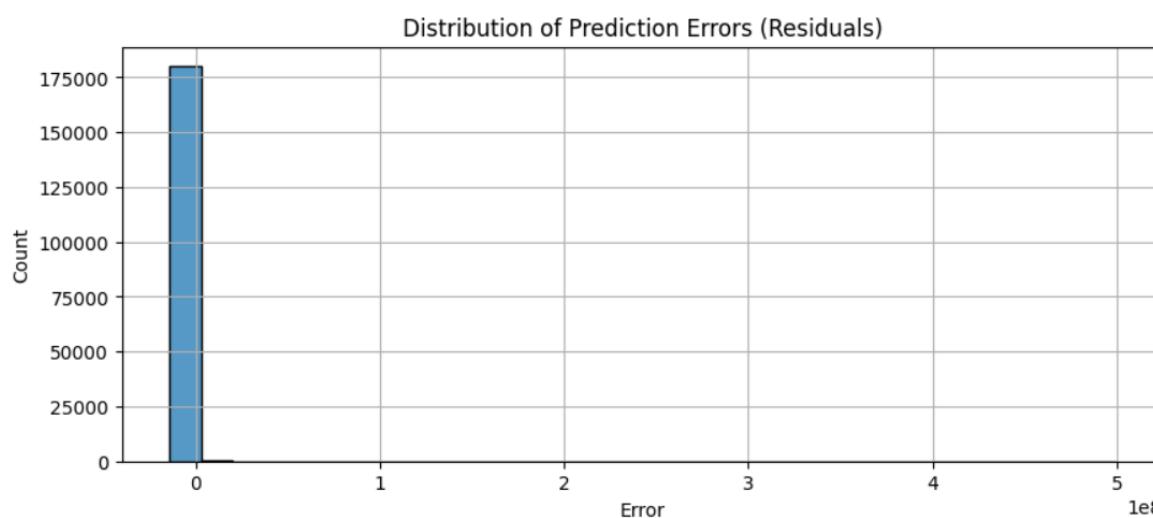
این نمودار خطای مقادیر پیش بینی($\text{residuals} = \text{Actual} - \text{Predicted}$) را نشان می دهد که بر روی محور y قرار گرفته اند و مقدار پیش بینی شده هم بر روی محور x قرار دارد. همانطور که مشاهده می شود خطاهای اکثر صفر و یا نزدیک به صفر هستند.

تحلیل : نمودار به شدت چپ چوله است . چپ چوله بودن نمودار یعنی میانگین دادهها از میانه و مد کمتر است. این به این دلیل است که دادههای بزرگتر (در سمت راست) میانگین را به سمت خود می کشند، در حالی که تعداد بیشتری داده در سمت چپ وجود دارد که باعث می شود مد نمودار به سمت چپ کشیده شود. اکثر خطاهای نزدیک صفر هستند اما تعداد کمی خطای خیلی

بزرگ وجود دارد که نشان می دهد در داده ها ، داده های پرت هم وجود دارند. در نتیجه ، مدل بر روی داده های نرمال خوب عمل می کند اما به داده های پرت حساس است.

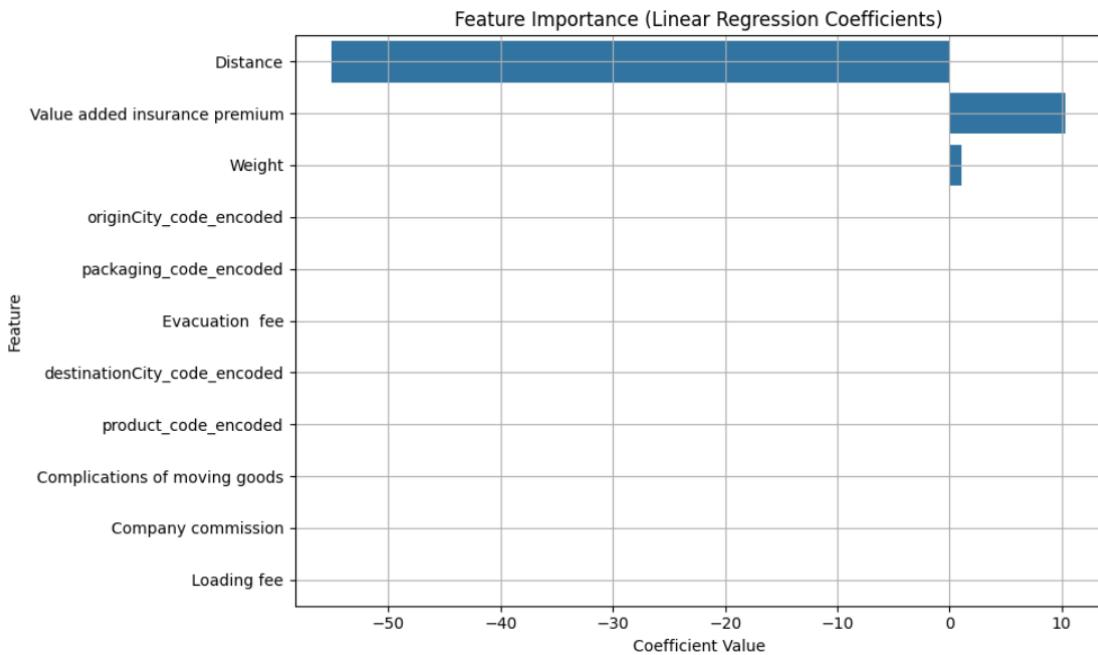
(Distribution of prediction errors)

نمودار هیستوگرام هم نشان دهنده ای این است که تعداد خیلی زیادی از داده ها حدود ۱۷۵۰۰۰ از خطاهای مقدار صفر داشته اند اما تعداد خیلی کمی از داده ها هم وجود دارند که خطایی نزدیک به صفر دارند.



(Feature Importance)

نمودار زیر ، ضرایب ویژگی ها در مدل رگرسیون خطی را نشان می دهد. مقادیر مثبت داری تاثیر مثبت بر روی مدل بوده اند و مقادیر منفی دارای تاثیر منفی روی مدل بوده اند.



تحلیل : مهمترین ویژگی در این مدل ، ویژگی مسافت (Distance) بوده است. و تاثیرش خیلی بوده است. و همینطور تاثیرش بشدت منفی بوده است. ویژگی تاثیرگذار دوم ، ویژگی حق بیمه ارزش افزود (Value added insurance premium) است که تاثیرش مثبت است. سومین ویژگی تاثیرگذار ، ویژگی وزن (Weight) است که دارای تاثیر مثبت است. بقیه ویژگی ها Company و Complications of moving goods و Loading fee و Evacuation fee و packaging_code_encoded و product_code_encoded و commission و destinationCity_code_encoded و originCity_code_encoded هیچ تاثیری بر روی مدل نداشته اند.

مدلسازی زمان رسیدن کالا با استفاده از مدل رگرسیون خطی

برای اینکه ما در دیتاست مربوط به حمل و نقل ، ستونی به نام زمان رسیدن کالا نداریم ، باید این ستون را خودمان بسازیم . این ستون را با استفاده از دو ستون تاریخ ارسال (Date of issue) و تاریخ تحویل کالا (Delivery date) ، ستون زمان رسیدن یا Delivery_days بدست می آوریم یعنی اختلاف آن دو ستون نام بردہ می شود ستون مدت زمان رسیدن کالا .

به علت اینکه ستون های تاریخ ارسال و تایخ تحویل کالا دیتاتایپ از نوع آبجکت دارند و همینطور تاریخ شمسی هستند ، پایتون از تاریخ شمسی پشتیبانی نمی کند برای همین باید در ابتدا تابعی برای تبدیل تاریخ شمسی به میلادی بنویسیم و بعد از آن ، این تابع را بر روی این دو ستون اعمال کنیم.

در اینجا ما کد برای تبدیل تاریخ شمسی به میلادی نوشته ایم که به صورت زیر است :

```
تابع تبدیل تاریخ شمسی به میلادی#
def shamsi_to_miladi(date_str):
    فرق بر اینکه که فرمت تاریخ به صورت :
    #1403/02/01
    y, m, d = map(int, date_str.split('/'))
    return jdatetime.date(y, m, d).togregorian()
```

البته برای اینکه بتوانیم این تابع را اجرا کنیم باید کتابخانه `jdatetime` نصب کنیم و سپس ایمپورت کنیم ؛ باید این کتابخانه را نصب کنیم تا بتوانیم تاریخ شمسی را به تاریخ میلادی برگردانیم زیرا برای تبدیل دیتاتایپ به تاریخ ، پانداس تاریخ شمسی پشتیبانی نمی کند. پس از اعمال این تابع روی دو ستون مذکور ، دیتاتایپ را به تاریخ و `datetime` تغییر می دهیم. سپس با یک خط کد زیر

```
ساخت ستون "مدت زمان رسیدن" بر حسب روز#
transport_df['Delivery_days']=(transport_df['Delivery date'] - transport_df['Date of issue']).dt.days
```

اختلاف دو ستون را بدست می آوریم و در ستون جدید به نام `Delivery_days` ذخیره می کنیم. برای اینکه مطمئن شویم مقادیر منفی در ستون ساخته شده (یعنی مدت زمان رسیدن کالا) وجود ندارد این خط کد را می زنیم :

```
▶ (transport_df['Delivery_days'] < 0).sum()
▶ np.int64(0)
```

همانطور که می بینید هیچ مقدار منفی در این ستون نداریم همانطور که انتظار می رفت اما برای اطمینان می خواستیم ببینیم تاریخ تحویل زودتر از تاریخ ارسال نباشد.

برای اینکه در داده هایمان نویز نداشته باشیم ، چون قبل از این خط کد یک خط کد زدیم تا ببینیم تعداد روزهای مدت زمان تحویل برای هر مقدار چقدر است مثلا یک مقدار داشتیم که ۴۶۰ روز بوده است که یکبار فقط اتفاق افتاده یا مثلا یه مقدار در ستون Delivery_days داشتیم که مقدارش ۲۳۳ روز بوده است که اینها ممکن است نویز باشد پس یک شرط می گذاریم :

```
transport_df1 = transport_df[transport_df['Delivery_days'] < 60]  
transport_df1
```

این خط می آید و مقادیری در ستون Delivery_days انتخاب می کند که مقدارشان از ۶۰ کمتر باشد یعنی مدت زمان رسیدن کالا به دست مشتری کمتر از ۶۰ باشد تا دیگر داده هایمان فاقد نویز باشند.

در این مرحله باید متغیر مستقل و متغیر وابسته و یا هدف را برای مدل مشخص کنیم. برای انتخاب متغیرهای مستقل مانند مدلسازی قبلی برای انتخاب برخی ویژگی های عددی که ماهیت کد دارند Destination و Code of the city of origin Target-Encoding انجام می دهیم. بر روی ستونهای city code انجام می دهیم. طبق کد روبرو این ویژگی ها را برای متغیر مستقل انتخاب می کنیم.

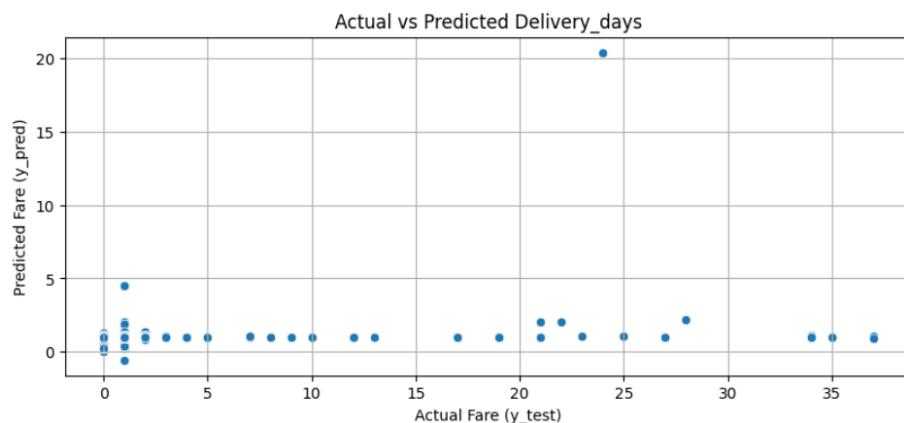
```
❸ #selecting features for independent variable  
selected_features1=[  
    'Weight',  
    'Distance',  
    #'Evacuation fee',  
    'Loading fee',  
    'Complications of moving goods',  
    'originCity_code_encoded',      #instead of Code of the city of origin  
    'destinationCity_code_encoded' #instead of Destination city code  
]  
  
X = transport_df1[selected_features1]
```

برای متغیر وابسته هم Delivery_days را قرار می دهیم زیرا می خواهیم همین ستون را پیش بینی کنیم. سپس طبق روال همیشگی مدل را به دو قسمت آموزش و تست تقسیم بندی می کنیم و ۳۰٪ داده ها را به بخش تست اختصاص می دهیم. بعد از آن مدل را می سازیم و فیت می

کنیم و پیش بینی می کنیم و مقادیر MAE و MSE و R2 Score را بدست می آوریم و در انتها نمودارهای مورد نیاز را رسم می کنیم.

معیار	مقدار بدست آمده	تحلیل
Mean Absolute Error	0.03576690191630671	
Mean Squared Error	0.11747461759097932	
R2 Score	0.036683123016134744	

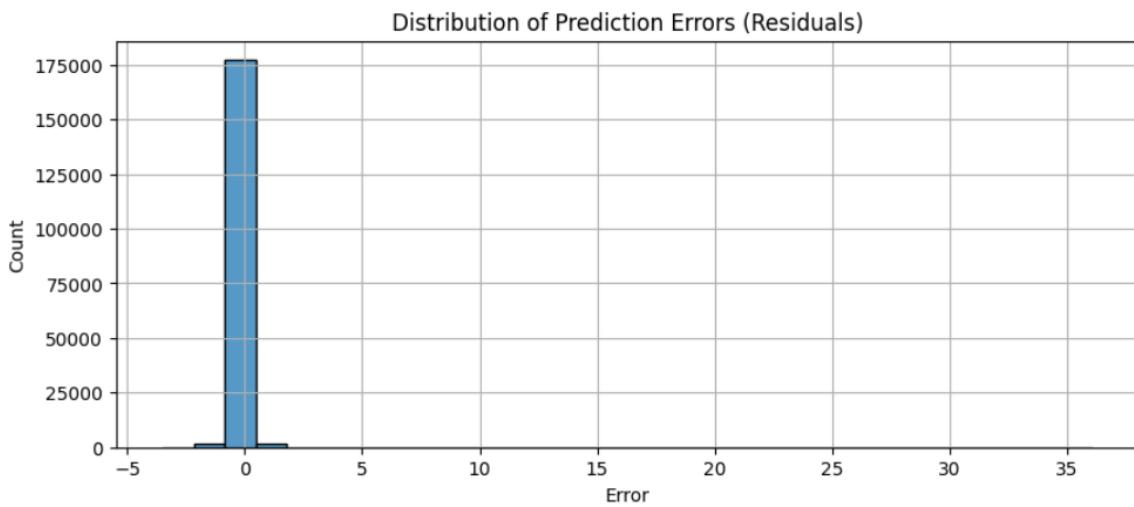
نمودار Actual vs Predicted Delivery_days



اکثر نقاط در بازه‌ی ۰ تا ۵ روز متتمرکزند. داده‌ها پراکندگی زیادی دارند، یعنی مدل همیشه دقیقاً در خط $y = x$ نیست و پیش‌بینی هم دقیق نیست. برای مقادیر بالاتر (مثلًاً بالای ۱۰ روز)، مدل یا اشتباه پیش‌بینی کرده یا داده‌ی کافی نداشته است.

در نتیجه، پیش‌بینی مدل برای زمان‌های کوتاه (زیر ۵ روز) نسبتاً بهتر عمل می‌کند. اما برای زمان‌های بلندتر تحویل، دقت مدل به وضوح افت می‌کند.

نمودار residuals vs y_pred



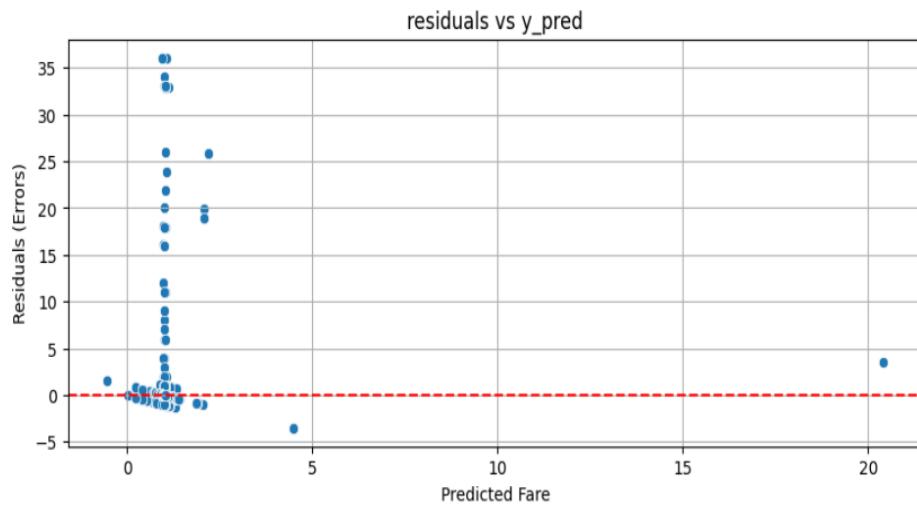
توزیع بسیار چگال در اطراف صفر است و اکثر خطاها کم‌اند. تعداد کمی داده با خطای بالا وجود دارد و احتمالاً مقادیر پرت یا موارد خاص هستند.

در نتیجه، مدل در مجموع عملکرد خوبی در بیشتر داده‌ها دارد، اما چند نمونه خاص وجود دارد که بسیار بد پیش‌بینی شده‌اند.

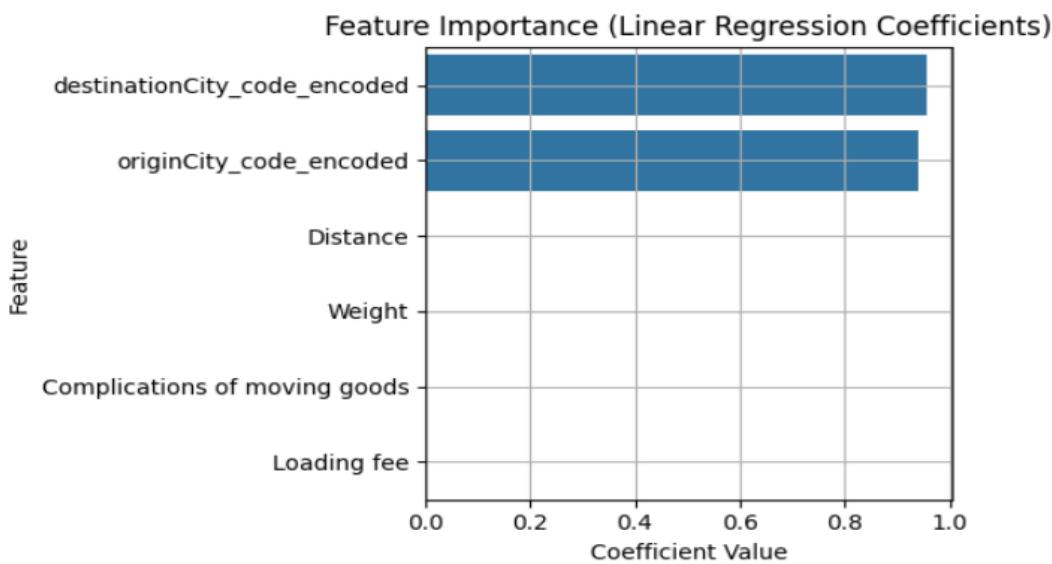
نمودار توزیع خطاها (Distribution of Prediction Errors)

اکثر باقیمانده‌ها (residuals) در محدوده نزدیک به ۰ هستند و یعنی بیشتر پیش‌بینی‌ها نسبتاً خوب بودند. اما چند مقدار پرت داریم که خطاها خیلی بالا یا پایین دارند. خط قرمز ($y=0$) نشونده‌نده خطای صفر است، و نقاط دور از آن نشان‌دهنده پیش‌بینی‌های ضعیفتر.

در نتیجه، مدل در بیشتر موارد خطای کمی دارد، ولی گاهی پیش‌بینی بهشدت اشتباه دارد (outliers).



نمودار اهمیت ویژگی ها



در این نمودار مشاهده می کنید که هر ویژگی چه ضریب اهمیتی در مدل رگرسیون خطی برای پیش بینی مبلغ بیمه دارد. در این نمودار تاثیر منفی مشاهده نمی شود و همه ی ویژگی ها یا تاثیر مثبت دارند و یا تاثیری ندارند (یعنی ضریب اهمیتشان صفر است). ویژگی های

دو میان تاثیر را دارند؛ تاثیر هر دو ویژگی نزدیک به ۱ است و این نشان دهنده اهمیت بالای این

`originCity_code_encoded` و `destinationCity_code_encoded`

ویژگی ها در این مدل می باشد. بقیه ۴ ویژگی دیگر تاثیرشان در این مدل صفر بوده و عملاً تاثیری نداشته اند.

به دلیل اینکه مقدار دقต در این مدل بسیار پایین است اگرچه مقدار MAE , MSE به نظر خوب هستند اما دقت بشدت پایین است پس چاره ای جز تعویض مدل پیش بینی نداریم.

مدل XGBoost Regression برای پیش بینی مبلغ بیمه

در ابتدا کتابخانه های مورد نیاز برای این مورد را ایمپورت می کنیم مانند

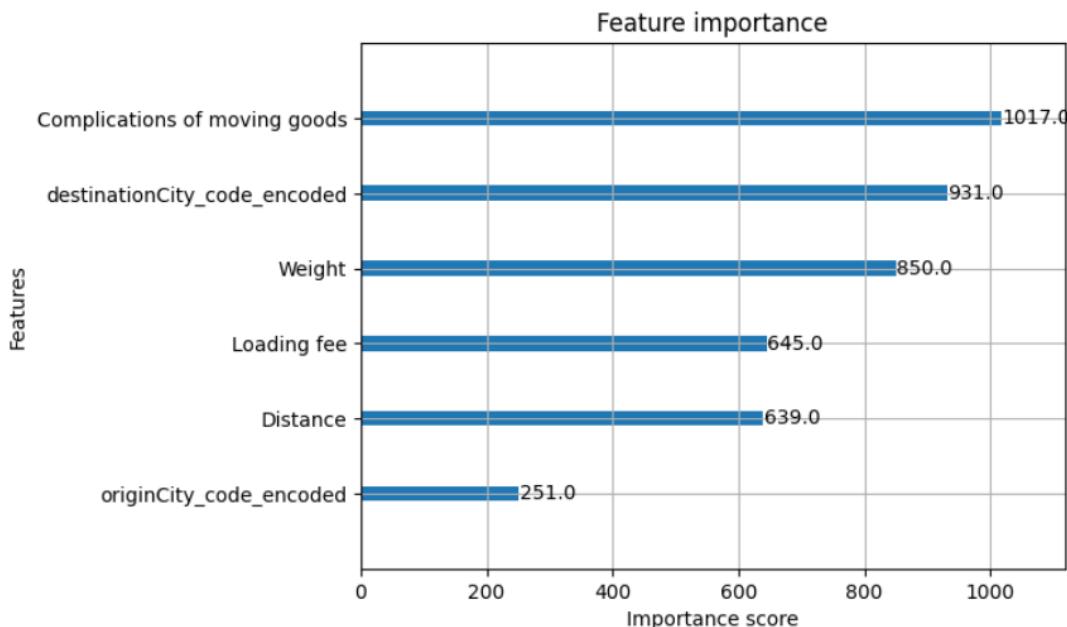
```
from xgboost import XGBRegressor
```

برای استفاده از مدل XGBoost سپس ویژگی های مدنظر را برای متغیر مستقل انتخاب می کنیم (چون می خواهیم همان هدف مدل قبلی را پیش بینی کنیم ویژگی هایی که برای همان مدل به عنوان متغیر مستقل انتخاب کردیم را درنظر می گیریم و برای دو تا از ویژگی ها نیاز با انجام Target-Encoding بود که در همان مدل قبلی این کار را انجام دادیم پس از همان استفاده می کنیم. برای متغیر وابسته و هدف هم ویژگی Delivery_days را انتخاب می کنیم. داده ها را به دو دسته آموزش (۷۰٪) و تست (۳۰٪) تقسیم می کنیم. مدل را می سازیم ، آموزش می دهیم ، فیت می کنیم و با استفاده از X_{test} مدل را پیش بینی می کنیم. مقادیر MAE ، MSE و R2 Score را محاسبه می کنیم. و در آخر نمودار اهمیت ویژگی را برای مدل XGBoost با استفاده از دستور `plot_importance()` از کتابخانه `xgboost` رسم می کنیم.

تحلیل	مقدار بدست آمده	معیار
مقدار بدست آمده برای میانگین قدر مطلق خطای مطلوب است اما چون دقت و مقدار بدست آمده برای R2-Score بسیار پایین است ، تاثیری ندارد.	0.03413015976548195	Mean Absolute Error

Mean Squared Error	0.11548195779323578	مقدار بدست امده برای مربع میانگین خطای بسیار خوب است اما چون دقت پایین است ، تاثیری ندارد.
R2 Score	0.053023338317871094	دقت بدست آمده از این مدل به شدت پایین است و اصلاً به این مدل نمی‌توان اعتماد کرد.

نمودار اهمیت ویژگی



در این مدل ، با توجه به نمودار اهمیت بالا مشاهده می شود که تاثیر ویژگی ها افزایش پیدا کرده است؛ برای مثال در مدل قبل ضریب اهمیت ۴ ویژگی آخر صفر بود اما در این مدل ضریب اهمیتشان بهبود قابل توجهی داشته اند.

در نتیجه ، استفاده از این مدل (تعویض مدل از مدل رگرسیون خطی به XGBoost Regression) هم تاثیری در بهبود و افزایش دقت نشده است. با تعویض مدل دقت از ۳٪ به ۵٪ افزایش پیدا کرده

است اما دقته که مدنظر ماست باید بیش از ۹۰٪ باشد. ممکن است با افزایش ویژگی، مدل بهبود پیدا کند؛ نرمالسازی داده ها هم به بهبود مدل کمک خواهد کرد.

بخش خوشه بندی ها

KMeans Clustering

برای اینکه بتوانیم راننده ها را خوشه بندی کنیم به یک ستونی نیاز داریم که مقادیرش برای هر راننده منحصر بفرد و یونیک باشد؛ ستونی که می توانیم براسی این کار استفاده کنیم، ستون زیرا مقادیر این ستون برای هر راننده دارای یک مقدار یکتاست زیرا کارت Driver smart card هوشمند راننده است.

کتابخانه های مورد نیاز برای خوشه بندی را ایمپورت می کنیم. سپس طبق تصویر زیر براساس

```
driver_stats = transport_df.groupby('Driver smart code').agg({
    'Delivery_days' : [ 'mean' , 'std' , 'count' ] ,
    'Weight' : 'mean' ,
    'Distance' : 'mean' ,
    'Total fare' : 'mean'
})
```

ستون Delivery_days تجمع انجام می دهیم و می خواهیم براساس همین ستون میانگین، انحراف معیار، تعداد را برای ستون Weight، میانگین ستون Distance، میانگین ستون Total fare و میانگین حالت را محاسبه

```
print(driver_stats.isnull().sum())
```

	Delivery_days	mean	0	print(driver_stats.isnull().sum())
	std	49172		محاسبه می کنیم ببینیم ستون های
	count	0		دیتافریممان مقادیر خالی دارد یا خیر.
Weight	mean	0		
Distance	mean	0		
Total fare	mean	0		
	dtype:	int64		

خروجی بدست آمده نشان می دهد که در ستون انحراف معیار Delivery_days ، ۴۹۱۷۲ مقدار NaN وجود دارد ، به این علت است که بعضی از راننده ها فقط یکبار حمل داشته اند (فقط یک رکورد در دیتاست به آنها اختصاص دارد). برای همین نمی توان انحراف معیار برای ستون Delivery_days بدست بیاورد و مقادیر را خالی برمیگرداند. برای حل این مشکل می توانیم مقادیر خالی را با صفر جایگزین کنیم این یعنی انحراف معیار مربوط به آن سطر صفر شده است.

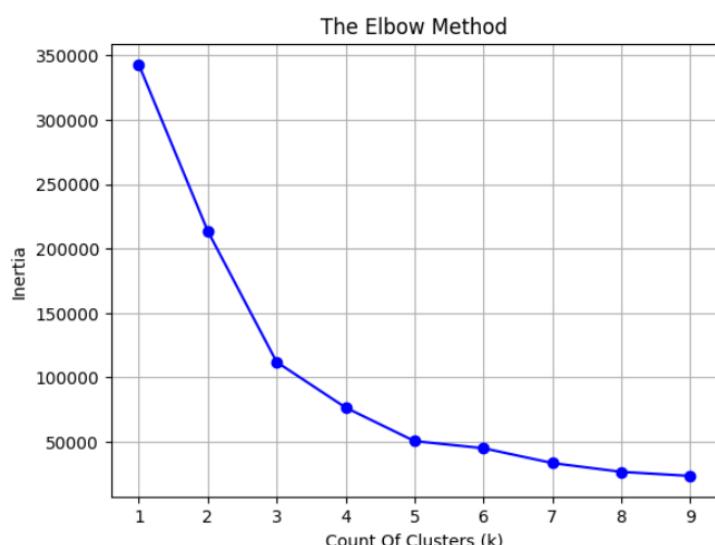
سپس نام ستون های دیتافریم driver_stats را تغییر می دهیم و نام های مناسب انتخاب می کنیم برای ستون ها .

```
#normalizing data
scaler=StandardScaler()
scaled_data=scaler.fit_transform(driver_stats)

چون می خواهیم نمودار رسم کنیم کاهش بعد انجام می دهیم#
کاهش بعد با روشن#
#PCA
pca=PCA(n_components=2)
pca_result=pca.fit_transform(scaled_data)
```

داده های دیتا فریم را نرمالسازی می کنیم. سپس چون می خواهیم نتیجه را روی نمودار مصورسازی کنیم از حالا کاهش بعد با روشن PCA انجام می دهیم.

با روش Elbow مقدار مناسب برای تعداد خوشه را بدست می آوریم یعنی مقدار مناسب برای k . حلقه ای برای اجرای KMeans با تعداد خوشه های مختلف (از ۱ تا ۱۰) و محاسبه inertia برای هر یک از آنها استفاده شده است.



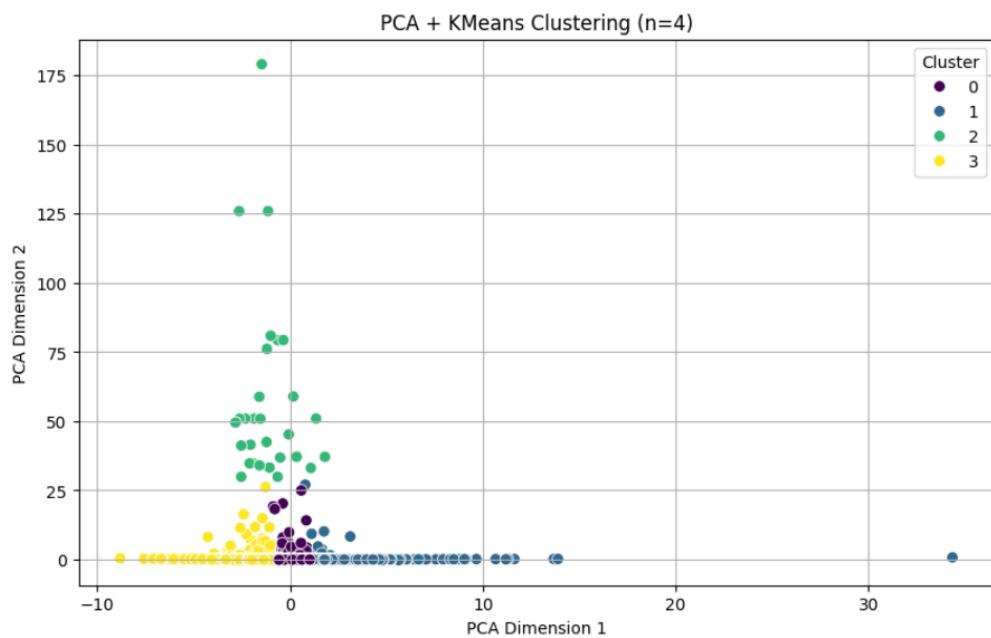
در نهایت با رسم نمودار Elbow ، که با `x= list(k_range) ، plot()` دستور است و `y=` که برای هر مقدار `k` در `k_range` inertia آن ذخیره شده است . این نمودار روبه رو ، نمودار Elbow است که می توان زانویی را بین مقادیر ۳ تا ۵ دید. مقدار

۴ بهترین مقدار برای زانویی است. پس تعداد خوشه ها برابر با ۴ خواهد بود.

```
#KMeans Clustering
kmeans=KMeans(n_clusters = 4 , random_state = 42)
clusters = kmeans.fit_predict(pca_result)
#scaled_data
#pca_result

رسم نمودار
#scatterplot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1], hue=clusters, palette='viridis', s=50)
plt.title('PCA + KMeans Clustering (n=4)')
plt.xlabel('PCA Dimension 1')
plt.ylabel('PCA Dimension 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```

در مدل KMeans ، تعداد خوشه ها را ۴ و رندوم استیت را ۴۲ قرار می دهیم. سپس داده های کاهش یافته به ۲ بعد را به مدل می دهیم تا فیت و پیش بینی شوند. سپس با استفاده از scatterplot() نمودار را رسم می کنیم . x را بعد اول pca_result و y را بعد دوم pca_result قرار می دهیم . و hue را هم مقدار بدست آمده یعنی خوشه ها را که clusters نام دارد ، می گذاریم تا خوشه بندی براساس این ستون روی نمودار ظاهر شود. خوشه بندی به صورت زیر است :



خوشه ها را به دیتافریم driver_stats اضافه می کنیم و نام ستون را cluster می گذاریم.

هم اکنون که خوشه بندی را انجام داده ایم ، باید تحلیل کنیم که هر خوشه چه ویژگی هایی دارد و کدام راننده ها در چه خوشه ای قرار گرفته اند.

از هر خوشه `head()` می گیریم تا بررسی انجام دهیم.

تحلیل ویژگی های آماری برای هر خوشه انجام می دهیم. یعنی براساس ستون خوشه تجمعی انجام می دهیم و میانگین می گیریم. به صورت زیر :

همانطور که می بینید براساس خوشه های ۰ و ۲ و ۳ میانگین ستون دیگر بدست آمده اند. براساس همین ویژگی های آماری بدست آمده می توانیم نام برای هر خوشه انتخاب کنیم و خوشه ها را از این به بعد با نامشان بشناسیم.

```
[ ] driver_stats.groupby('cluster').mean()
```

cluster	avg_delivery_days	std_delivery_days	num_deliveries	avg_weight	avg_distance	total_fare
0	1.006064	0.024553	3.268108	14550.533739	676.210673	1.203791e+08
1	1.000644	0.020591	2.785393	20581.268046	1084.375741	2.245562e+08
2	148.469037	63.120845	3.781250	12940.526008	595.361952	1.015830e+08
3	1.014574	0.033089	11.108887	6061.771971	402.419111	5.469187e+07

می خواهیم با تفسیر جدول بالا نام های مناسبی برای هر خوشه انتخاب کنیم :

مطابق با جدول بالا برای اینکه بتوانیم نام مناسبی به هر خوشه بدھیم برچسبی از ۱ تا ۴ به هر سلول از جدول می دهیم تا بتوانیم نام انتخاب کنیم.

- مقدار ۱ در جدول پایین یعنی : زیاد
- مقدار ۲ در جدول یعنی : نسبتاً زیاد (متوسط)
- مقدار ۳ در جدول یعنی : نسبتاً کم (متوسط)
- مقدار ۴ در جدول یعنی : کم

cluster	avg_delivery_days	std_delivery_days	num_deliveries	avg_weight	avg_distance	total_fare
0	۳	۳	۳	۲	۲	۲
1	۴	۴	۴	۱	۱	۱
2	۱	۱	۲	۳	۳	۳
3	۲	۲	۱	۴	۴	۴

مثلا برای ستون avg_delivery_days مقدار ۳ برای خوشه ۰ یعنی این مقدار نسبتاً متوسط است

با توجه به جدول ویژگی های آماری . برای خوشه ۱ ، مقدار ۴ قرار دادیم یعنی مقدار میانگین

مدت زمان تحویل کم است. و بقیه سطرها و ستون ها هم به همین ترتیب با توجه به جدول ویژگی های بدست آمده . (این جدول بالا بیشتر به خودم برای نامگذاری کمک کرده است شاید توضیحاتم مربوط به جدول بالا زیاد خوب نباشد).

حال به کمک جدول بالا نام گذاری و برچسب دهی به خوشه ها را آغاز می کنیم .

- خوشه ۰ : 'راننده متعادل و نرمال'
- خوشه ۱ : 'راننده های سریع و سنگین بار'
- خوشه ۲ : 'راننده ها خیلی کند و پراکنده'
- خوشه ۳ : 'راننده های سریع و پرکار (سبک بار)'

```
[ ] [برچسب گذاری خوشه ها]
label_map = {
    0 : 'Balanced drivers' , #راننده متعادل و نرمال
    1 : 'Fast heavy-load drivers' , #راننده های سریع و سنگین بار
    2 : 'Very slow and scattered drivers' , #راننده ها خیلی کند و پراکنده
    3 : 'Fast frequent light-load drivers' #راننده های سریع و پرکار (سبک بار)
}

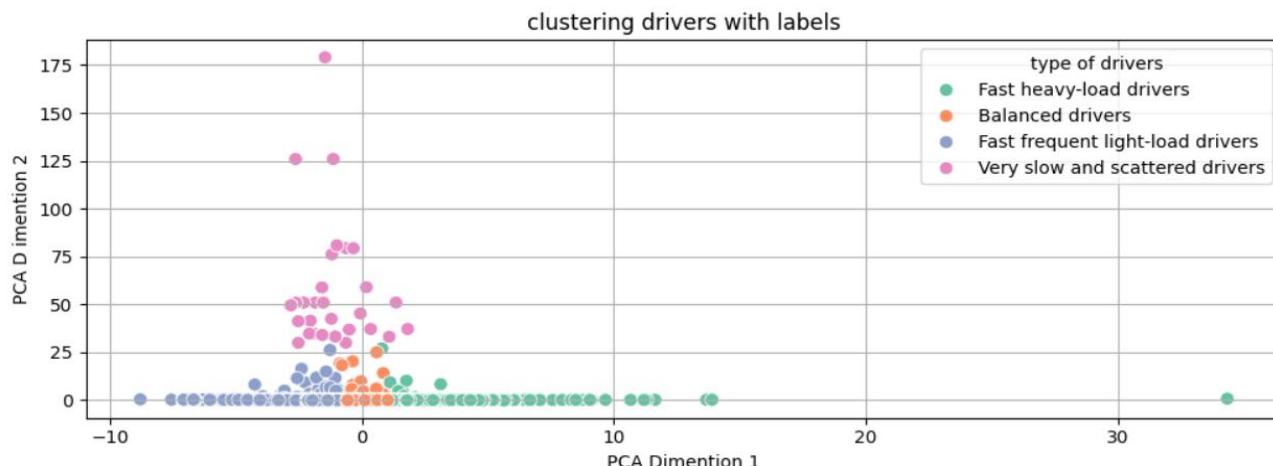
# اضافه کردن برچسب ها به دیتا فریم
driver_stats['cluster_label'] = driver_stats['cluster'].map(label_map)
```

یک دیکشنری است که برای · نام Balanced drivers و برای خوشه ۱ نام Fast label_map می گذاریم برای بقیه خوشه ها هم به همین صورت . سپس با استفاده ازتابع map() که دیکشنری ساخته شده را به آن پا داده ایم می آید نام های تعریف شده را به خوشه ی driver_stats ستون را cluster_label می نامیم و به دیتا فریم مناسبش تطبیق می دهد. و این ستون را pca_result می سازیم و دیتا فریم اضافه می کنیم. با توجه به کدهای بالا یک دیتا فریم جدید می سازیم و دیتا فریم را در

```
# ساخت دیتا فریم از نتایج PCA و اضافه کردن ستون برچسب ها
pca_df = pd.DataFrame(pca_result, columns=['PCA1','PCA2'] , index=driver_stats.index)
pca_df['cluster_label'] = driver_stats['cluster_label']
```

آن قرار می دهیم و دو ستونی که در این دیتا فریم (pca_result) هست را نامگذاری جدید انجام بدیم به نامهای PCA1 , PCA2 و ایندکس هم همان ایندکس pca_result یعنی smart card قرار می دهیم. ستون cluster_label را هم به این دیتا فریم جدید می افزاییم.

در آخر نمودار را براساس دیتا فریم pca_df رسم می کنیم و خوشه ها نمایش می دهیم. در قسمت hue این دفعه ستون cluster_label قرار می دهیم تا براساس این ستون خوشه بندی انجام شود البته تفاوتی با ستون خوشه نمی کند. تنها تفاوت این است که در قسمت legend در نمودار نام های هر خوشه مشخص است و کاربر یا هر شخصی که نمودار را ببیند درک بهتری از نمودار خواهد داشت برخلاف زمانیکه در قسمت legend فقط خوشه ها از ۰ تا ۳ نمایش داده می شد. در نهایت نمودار به صورت زیر خواهد بود :



خوشه بندی شرکت ها براساس رفتار هزینه ای با روش DBSCAN

تمام ستون های هزینه ای طبق کد روبرو را برای خوشه بندی شرکت ها انتخاب می کنیم.

```
انتخاب فیچرهای مرتبط با رفتار هزینه ای#
features = [
    'Total fare',
    'Received from driver',
    'Company commission',
    'Insurance amount',
    'Loading fee',
    'Evacuation fee',
    'Weight',
    'Distance',
    'Calculateion fare',
    'rent',
    'Value added insurance premium'
]
company_stats = transport_df[features]
```

و در دیتافریم جدیدی به نام company_stats ذخیره می کنیم. سپس براساس company code که ستونی در دیتابیس اصلی مان است، از ویژگی ها میانگین company_stats میگیریم. و در همین دیتافریم company ذخیره می کنیم. ایندکس این دیتافریم هم code است.

چک می کنیم ببینیم آیا مقدار خالی در دیتافریم وجود دارد یا خیر.



```
# گروهبندی روی شرکت ها و محاسبه میانگین
company_stats = transport_df.groupby('Company code')[features].mean()
company_stats.isnull().sum()
```



	0
Total fare	0
Received from driver	0
Company commission	0
Insurance amount	0
Loading fee	0
Evacuation fee	0
Weight	0
Distance	0
Calculateion fare	0
rent	0
Value added insurance premium	0

dtype: int64

نام ستون ها را عوض می کنیم چون میانگین گرفتیم طبق کد بالا، قبل اسم هر ستون mean اضافه می کنیم تا شفاف شود.

در مدل DBSCAN دو پارامتر داریم که باید مقداردهی شوند. یکی از این پارامترها نامش Epsilon است که نشان دهنده شعاع همسایگی است. دیگری min_sample که نشان دهنده حداقل تعداد همسایه هاست. باید مقدار eps با K-Distance Graph مشخص کنیم تا بهترین مقدار باشد. مقدار Epsilon مشخص می کند چند نقطه در همسایگی یک نقطه وجود دارند تا بتوانیم آن نقطه را جزو خوشه بدانیم. قبل از انجام

مرحله بالا ، اول بر روی دیتافریم نرمالسازی انجام می دهیم تا تفاوت داده ها مشکل ساز نشود. DBSCAN به فاصله ها حساس است، پس قبل از هر کاری داده ها با StandardScaler استاندارد شده اند.

حال الگوریتم k-نزدیکترین همسایه را بر روی داده های نرمالسازی شده یعنی scaled_data فیت می کنیم. تعداد همسایه ها را ۳ قرار داده ایم. یعنی اینجا n_neighbors=3 یعنی برای هر نقطه، فاصله تا سه نزدیکترین نقطه محاسبه شده است.

این خط distances, indices = nbrs.kneighbors(scaled_data) برای هر داده در دیتافریم،

```
پیدا کردن مقدار مناسب #Epsilon
#K-distance graph
neigh = NearestNeighbors(n_neighbors=3)
nbrs = neigh.fit(scaled_data)
distances, indices = nbrs.kneighbors(scaled_data)

# مرتبسازی فاصله ها برای ترسیم
distances = np.sort(distances[:, 2])
plt.figure(figsize=(8, 4))
plt.plot(distances)
plt.title("K-distance graph for DBSCAN (k=4)")
plt.xlabel("Data points sorted by distance")
plt.ylabel("4th Nearest Neighbor Distance")
plt.grid(True)
plt.show()
```

فاصله اش تا نزدیکترین همسایه ها را محاسبه می کند. nbrs.kneighbors() برای هر سطر در دیتافریم، k تا نزدیکترین همسایه را پیدا می کند و فاصله آنها (در فضای ویژگی ها) و ایندکس (index) آن همسایه ها را بر می گرداند. خروجی اینتابع برای distances یک آرایه دو بعدی است که در هر ردیف، فاصله تا k تا نزدیکترین همسایه نوشته شده است.

خروجی اینتابع برای indexes هم یک آرایه دو بعدی است که آرایه دو بعدی دو بعدی است که در هر ردیف، ایندکس (شماره سطر) k تا همسایه های نزدیک تر برای هر سطر را می دهد.

فاصله سطر 0 با خودش، و با دو تا همسایه دیگه #

سطر 0 با خودش، با سطر 12 و 34 نزدیک #

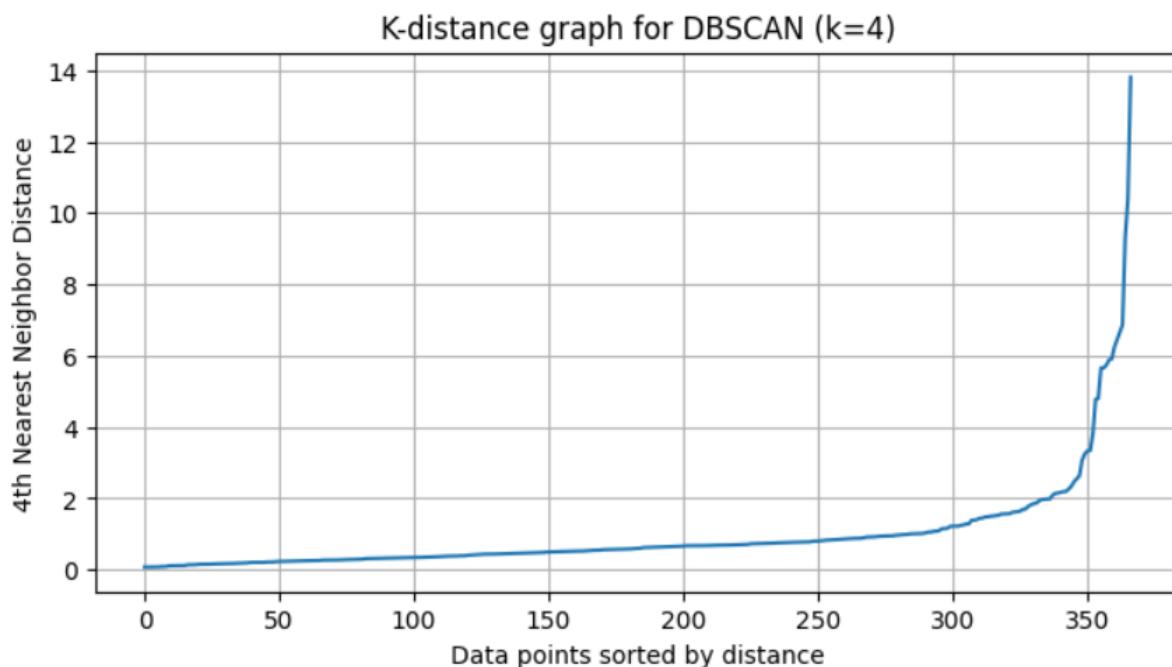
برای DBSCAN، باید متوجه شویم هر نقطه چقدر به بقیه نزدیک است. پس با استفاده از این فاصله ها

می توانیم نقطه شکست (elbow point) در K-distance graph را رسم کنیم و مقدار مناسب eps را انتخاب کنیم.

یعنی از بین ۳ فاصله (چون $n_{\text{neighbors}}=3$) فقط فاصله‌ی سومین نزدیک ترین همسایه را برمی‌داریم. سپس این فاصله‌ها را صعودی مرتب می‌کنیم.

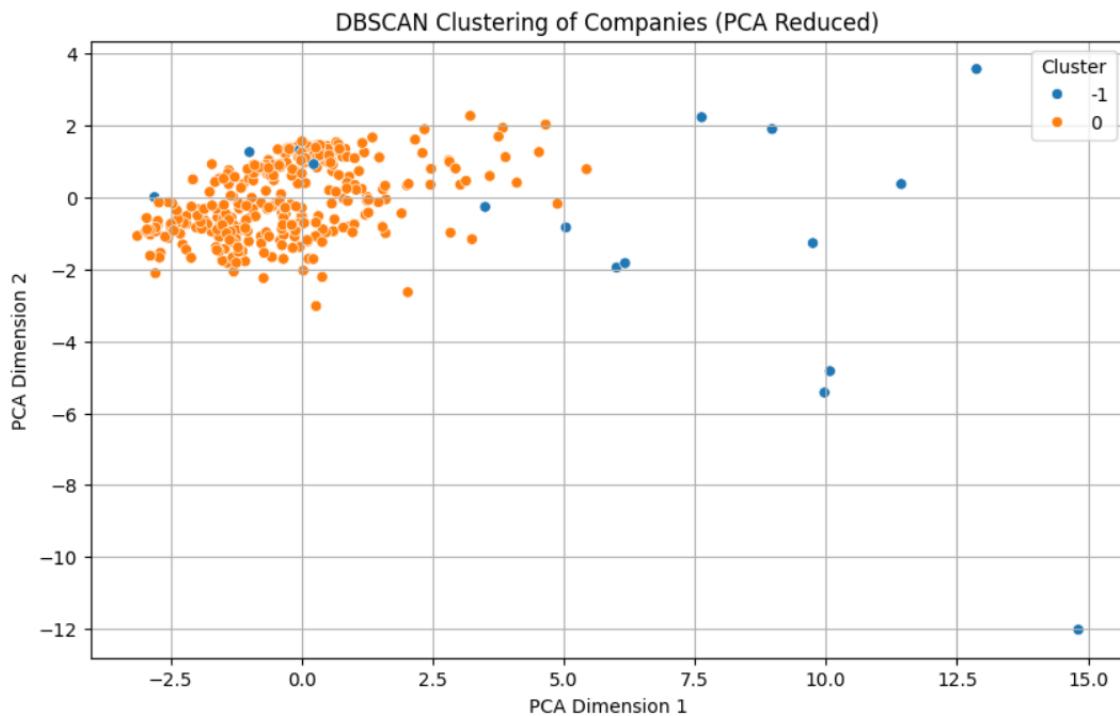
همین distance را بر روی نمودار رسم می‌کنیم تا بتوانیم بهترین مقدار برای اپسیلون را پیدا کنیم.

طبق نمودار بدست آمده، اگر ارقام محور y را نگاه کنیم می‌توانیم متوجه شویم که نمودار از کدام قسمت به بعد شیب ناگهانی داشته است. با توجه به شکل در $y=2$ نمودار یک شیب ملایمی گرفته اما از $y=3$ به بعد شیب نمودار تندمی شود و از نقطه $y=4$ شیب خیلی خیلی تند می‌شود و نشان دهنده داده‌های خیلی دور افتاده و پرت هستند. در نهایت با توجه به نمودار مقدار مناسب برای epsilon برابر است با ۳. این مقدار باعث می‌شود اکثر نقاط نرمال در خوش‌ها قرار بگیرند و نقاطی که فاصله‌ی زیادی با بقیه دارند به عنوان نویز شناسایی شوند.



اکنون الگوریتم DBSCAN را پیاده سازی می کنیم. و ستون cluster را به دیتافریم company_stats اضافه می نماییم . و ایندکس این دیتافریم که براساس کد شرکت بود را ریست می کنیم و شماره قرار می گیرد. ابعاد دیتافریم را با استفاده از روش pca ، کاهش می دهیم و به ۲ بعد تبدیل میکنیم و سپس نمودار آن را رسم می کنیم.

همنطور که در نمودار زیر می بینید تعداد خوشه ها فقط ۱ عدد است یعنی فقط ۱ خوشه داریم که خوشه صفر است و خوشه -۱ هم نشان دهنده ی نویزها هستند. با توجه به اینکه فقط یک خوشه با استفاده از این روش خوشه بندی بدست آمد، میتوان گفت این روش به ما کمکی نکرد چون ما میخواهیم خوشه بندی انجام بدهیم تا ویژگی هر خوشه را بررسی کنیم.



پس با این وضعیت ، ما مجبوریم مدل خوشه بندی را عوض کنیم. و برای خوشه بندی شرکتها از روش Gaussian Mixture Model (GMM) استفاده کنیم

مدل Gaussian Mixture Model (GMM) برای خوشه بندی شرکت ها

توضیح کلی درباره مدل آمیخته گوسی در خوشه بندی می توان گفت : همانطور که مشخص است، در این شیوه خوشه بندی، فرض بر این است که هر خوشه از داده هایی با توزیع نرمال (گوسی) تشکیل شده و در حالت کلی نیز داده ها نمونه ای از توزیع آمیخته نرمال هستند. هدف از خوشه بندی مدل آمیخته گوسی یا نرمال، برآورد پارامترهای توزیع هر یک از خوشه ها و تعیین برچسب برای مشاهدات است. به این ترتیب مشخص می شود که هر مشاهده به کدام خوشه تعلق دارد.

در مدل DBSCAN فیچرهای مناسب را انتخاب کردیم و گروه بندی براساس شرکت ها را هم انجام دادیم . نرمال سازی داده ها را هم در آنجا انجام دادیم . پس هم اکنون به مرحله پیاده سازی الگوریتم و مدل می رسیم. قبل از پیاده سازی مدل ، با استفاده از دو معیار AIC ، BIC می توانیم تعداد خوشه های مناسب را برای مدل تشخیص دهیم.

به عنوان یک شاخص سنجش و انتخاب مدل مناسب، معیار ارزیابی AIC ، میزان اطلاعاتی که توسط مدل از دست می رود را اندازه گیری می کند. به این ترتیب AIC یک تعادل بین تعداد پارامترهای مدل (پیچیدگی مدل) و میزان برازش مدل روی داده ها ارائه می کند. با در نظر گرفتن این موضوع می توان گفت مدلی که توسط AIC مدل مناسب تشخیص داده شود، نه دارای بیش برازش (Overfitting) است و نه از کم برازش (Underfitting) رنج می برد و می توان آن را مدلی با برازش مناسب در نظر گرفت. معیار ارزیابی BIC به مانند معیار AIC ، نمایانگر میزان اطلاعاتی است که توسط مدل از دست رفته است و در نتیجه هر چه مقدار معیار ارزیابی BIC کوچکتر باشد، مدل مورد نظر نسبت به بقیه مدل ها، بهتر و مناسب تر است. از کتابخانه sklearn.mixture مدل GaussianMixture را ایمپورت می کنیم. سپس اعدادی که می خواهیم برای تعداد خوشه تست کنیم را با تابع range انتخاب می کنیم و در متغیر n_components_range می ریزیم. سپس دو آرایه خالی به نام های bics, aics ایجاد می کنیم تا مقادیر بدست آمده از مدل را در این دو ذخیره کند.

```

▶ from sklearn.mixture import GaussianMixture
    import matplotlib.pyplot as plt

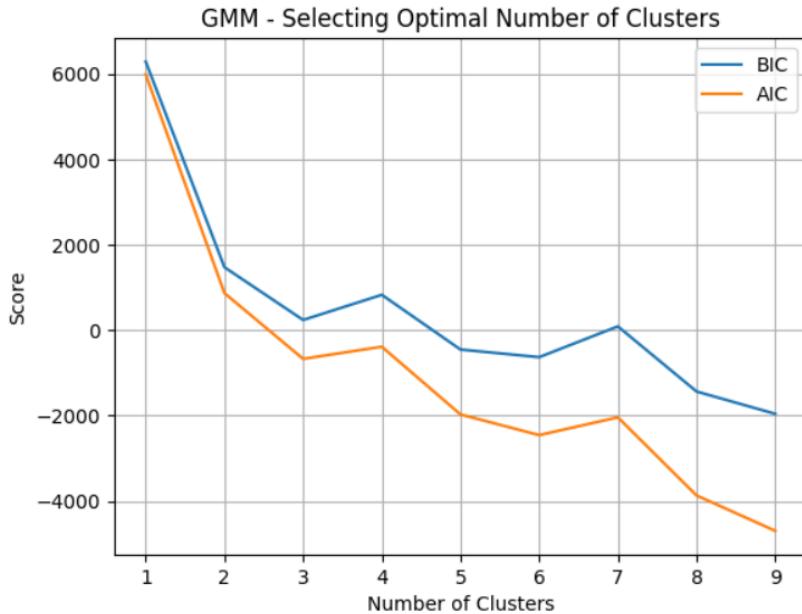
    از 1 تا 10 تعداد خوشه هایمان است که میخواهیم تست کنیم
    n_components_range = range(1, 10)
    # BIC (Bayesian Information Criterion)
    bics = []
    # AIC (Akaike Information Criterion)
    aics = []

    for n in n_components_range:
        gmm = GaussianMixture(n_components=n, random_state=42)
        gmm.fit(scaled_data)
        bics.append(gmm.bic(scaled_data))
        aics.append(gmm.aic(scaled_data))

    # رسم نمودار
    plt.plot(n_components_range, bics, label='BIC')
    plt.plot(n_components_range, aics, label='AIC')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Score')
    plt.title('GMM - Selecting Optimal Number of Clusters')
    plt.legend()
    plt.grid(True)
    plt.show()

```

یک حلقه ایجاد می کنیم تا مدل Gaussian Mixture Model تعداد خوشه های 1 تا 9 که در n_components_range متغیر قرار دادیم اجرا شود. و معیار های aic , bic را با دستورات



bics.append(gmm.bic(scaled_data)) و aics.append(gmm.aic(scaled_data)) برای هر خوشه از 1 تا 9 بدست می آوریم . سپس نمودار این دو معیار را رسم می کنیم. از نمودار () plot()

برای رسم استفاده می کنیم و مقدار $n_components_range$ را مقدار aic و مقدار bic را معيار های aic و bic قرار می دهیم. نمودار به صورت زیر خواهد بود :

هر دو خط aic و bic از ۱ تا ۹ خوش را نشان می دهد. طبق قانون کلی ، هر چقدر مقدار aic کمتر باشد بهتر است . ولی فقط نباید به کمترین مقدار نگاه کنیم ، بلکه باید به نقطه ای توجه کنیم که از آن به بعد بهبود خیلی کمتری رخ داده است که به این نقطه زانویی می گویند. هر دو نمودار پایین ترین نقطه آنها عدد ۹ است اما در بازه ۳ تا ۵ بهبود خیلی کمی رخ داده است پس می توانیم نقطه های ۳ تا ۵ را برای تعداد خوش انتخاب کنیم. در اینجا ما عدد ۵ را به عنوان تعداد خوش درنظر می گیریم. سپس مدل GMM را پیاده سازی می کنیم. سپس خروجی پیش بینی شده از مدل gmm را نام `gmm_clusters_company` ذخیره می کنیم ؛ این خروجی بدست آمده همان خوش بندی است و باید در ستون جدید در دیتابیس ذخیره شود. پس از انجام این کار ، ابعاد دیتابیس را به ۲ بعد کاهش می دهیم تا بتوانیم نمودارش را رسم کنیم.



ویژگی آماری هر خوش را بدست می آوریم :

```
company_stats_cluster_mean = company_stats.groupby('gmm_clusters_company').mean()
company_stats_cluster_mean
```

براساس ستون (تجمیع انجام می دهیم) gmm_clusters_company ، که ستون خوشه های بدست آمده از مدل هست ، از ستون های دیگر میانگین می گیریم و ویژگی آماری برای هر خوشه را بدست می آوریم که جدولی مانند زیر بدست می آید :

	Company code	Mean Total fare	Mean Received from driver	Mean Company commission	Mean Insurance amount	Mean Loading fee	Mean Evacuation fee	Mean Weight	Mean Distance	Mean Calculation fare	Mean rent	Mean Value added insurance premium	Cluster
gmm_clusters_company													
0	4.074193e+08	9.264228e+07	8.923439e+06	4.155070e+06	1.022575e+06	8.391293e+05	774931.946958	11158.527669	493.966890	8.230642e+07	7.554219e+07	8.353812e+04	-0.046154
1	6.505750e+08	1.131410e+08	2.169158e+07	8.562638e+06	1.762982e+06	3.255694e+06	191905.929897	3634.758356	1142.466037	9.159389e+07	9.046835e+07	1.602539e+05	-0.013333
2	1.111001e+08	3.852668e+08	7.435009e+07	2.227556e+07	5.366818e+07	4.545455e+05	0.000000	28909.090909	227.636364	2.784445e+08	2.784445e+08	5.366818e+06	-1.000000
3	3.961488e+08	1.128105e+08	1.096377e+07	5.881370e+06	1.160927e+06	8.900898e+05	76.473071	11801.697894	566.552520	9.766636e+07	9.753418e+07	9.235896e+04	-0.004630
4	2.727005e+08	4.427222e+08	4.620776e+07	2.282321e+07	1.099987e+07	2.080345e+07	0.000000	23921.500000	570.016667	3.586756e+08	3.586756e+08	9.704867e+05	-1.000000

براساس این جدول می توانیم برای هر خوشه نام و برچسب انتخاب کنیم و به دیتا فریم مربوطه اضافه کنیم. برای نامگذاری خوشه ها ، برای اینکه متوجه شویم کدام شرکت کالاهای سنگین وزن حمل می کند به ستون میانگین وزن (Mean Weight) در جدول بالا (جدول ویژگی آماری برای هر خوشه) نگاه می کنیم. برای مثال خوشه ۰ کالاهای سبک حمل می کند. خوشه ۱ هم کالاهای سبک حمل می کند اما خوشه ۲ کالاهای سنگین حمل می کند و خوشه ۳ کالاهای نسبتا سبک و خوشه ۴ کالاهای نسبتا سنگین حمل می کند.

برای اینکه رفتار هزینه ای شرکت ها را متوجه شویم به میانگین ستون های , Mean total fare توجه می کنیم . برای درآمد به میانگین Mean insurance amount , Mean evacuation fee , دو ستون Mean company commission, Mean received from driver توجه می کنیم. در نتیجه ای بررسی های انجام شده ، برچسب ها برای هر خوشه به صورت زیر هستند :

- خوشه ۰ : هزینه متعادل - سبک بار - مسافت طولانی - درآمد معمولی
- خوشه ۱ : درآمد بالا - هزینه متعادل رو به زیاد - سنگین بار - مسافت کم
- خوشه ۲ : درآمد کم - هزینه کم - باری و مسافت نسبتا کم
- خوشه ۳ : درآمد بالا - پرهزینه - باری و مسافت نسبتا زیاد

این لیبل ها و برچسب ها طولانی هستند در حالی که برچسب ها باید مفید و مختصر باشند پس می توانیم برای آنها این برچسب ها را به انگلیسی تعریف کنیم :

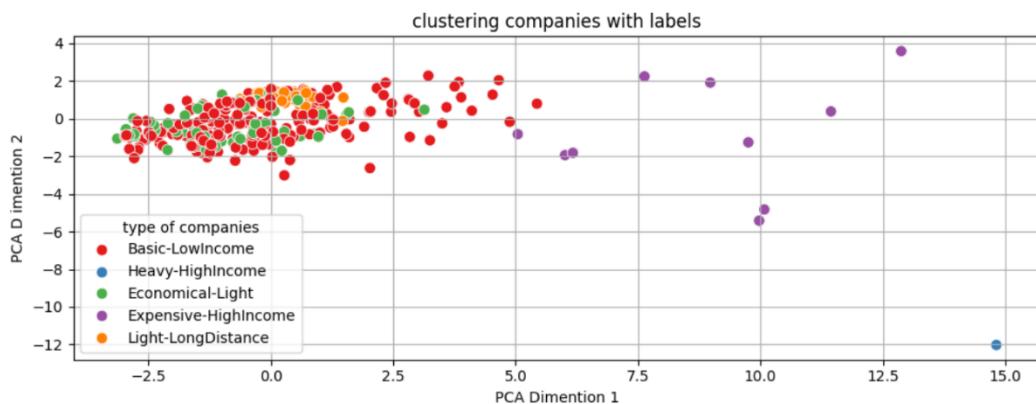
```
▶ #برچسب گذاری خوشة ها
company_label_map = {
    0 : 'Economical-Light' ,           #اقتصادی سبک
    1 : 'Light-LongDistance' ,          #سبک-مسافت بلند
    2 : 'Heavy-HighIncome' ,            #سنگین بار پُردرآمد
    3 : 'Basic-LowIncome' ,             #پایه ای کم درآمد
    4 : 'Expensive-HighIncome'        #پُرهزینه پُردرآمد
}

#اضافه کردن برچسب ها به دیتافریم
company_stats['gmm_clusters_company'] = company_stats['gmm_clusters_company'].astype(int)
company_stats['company_label_map'] = company_stats['gmm_clusters_company'].map(company_label_map)
company_stats
```

سپس این لیبل ها را به دیتافریم company_stats می افزاییم به کمک تابع map() تطبیق لیبل ها را به خوشخ ها انجام می دهیم.

سپس نمودار جدیدی براساس لیبلها رسم می کنیم تا واضح و شفاف باشد. همانند مدل قبلی در این مرحله ، دیتافریم جدید از pca_result1 می سازیم و دو ستون را نامگذاری می کنیم pca1_df([columns=['PCA1', 'PCA2']) و دو ستون خوشه و لیبل ها را هم به این دیتافریم که name دارد می افزاییم .

x ='PCA1' , y = 'PCA2' , hue = مقدار نمودار رسم قرار می دهیم و در بخش راهنمای برچسبها ظاهر می شوند و رنگنده خوشه ها هم انجام می گیرد.



همانطور که در خوشة Heavy_HighIncome در نمودار زیر مشاهده می کنید فقط یک نمونه وجود دارد که در قسمت پایین سمت راست نمودار قرار دارد. ما میتوانستیم خوشه بندی را با ۴ خوشه انجام دهیم تا این نمونه هم در خوشه Expensive-HighIncome قرار گیرد اما همانطور که مشاهده می کنید تفاوت زیادی بین نمونه های دو دسته وجود دارد و بهترین مقدار برای تعداد خوشه ها همان ۵ تاست. اگر ۴ خوشه می گرفتیم ویژگی های نمونه ای که در خوشه Heavy_HighIncome نادیده گرفته می شد.

خوشه بندی مسیرها براساس تاخیر ، مسافت و کرایه با استفاده از روش KMeans Clustering

با توجه به سوال مطرح شده برای خوشه بندی ، ما باید ستونی به نام تاریخ و زمان برنامه ریزی شده برای تحويل کالا داشته باشیم تا بتوانیم با محاسبه اختلاف این ستون و ستون تاریخ تحويل ستون تاخیر را ایجاد کنی ؛ اما ما در این دیتاست همچین ستونی نداریم برای همین این خوشه بندی را براساس دو ستون مسافت و کرایه انجام می دهیم.

دو ویژگی و ستون مسافت (Distance) و کرایه (Calculateion fare)(از بین کرایه ها برای خوشه بندی ما این کرایه را انتخاب می کنیم) را انتخاب و در دیتافریم جدیدی به نام path_df ذخیره می کنیم. سپس ستون مسیر را از اجتماع دو ستون نام شهر مبدأ و نام شهر مقصد در دیتاست اصلی می کنیم. بدست آمده به دیتافریم path_df (transport_df) می افزاییم.



باید ستون مسیر را بسازیم

```
path_df['path_col'] = transport_df['Name of the city of origin'] + ' → ' + transport_df['The name of the destination city']
path_df.head(20)
```

حال باید میانگین ویژگی ها را برای هر مسیر حساب کنیم. براساس ستون مسیر یعنی path_col گروپ بای می زنیم و از دو ستون دیگر میانگین می گیریم. و از این پس ایندکس های دیتافریم جدید یعنی path_stats براساس ستون path_col هستند.

```
حال باید میانگین ویژگی ها را برای هر مسیر حساب کنیم#
path_stats = path_df.groupby('path_col')[features1].mean()
path_stats
```

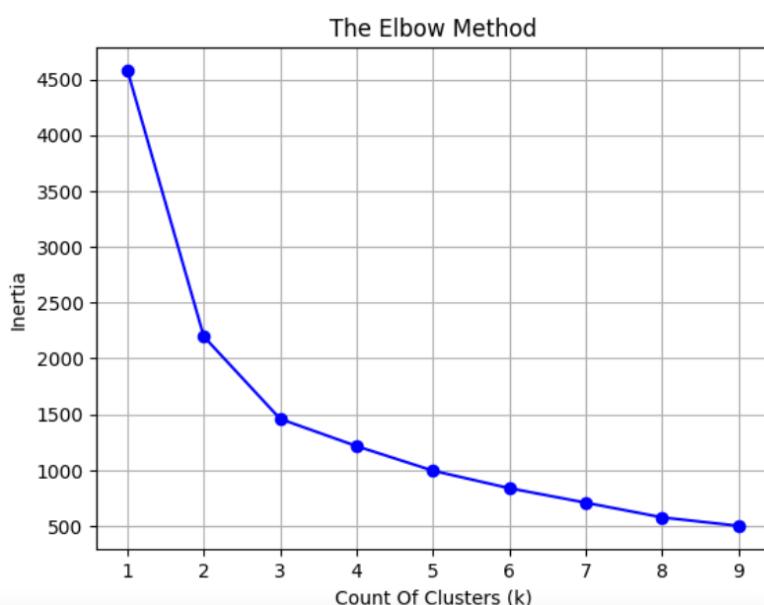
برای ستون ها مسافت و کرایه نام جدید می گذاریم :

```
نامگذاری جدید برای ستون های دیتاست#
#path_stats
path_stats.columns = [
    'path_col',
    'Mean Distance',
    'Mean Calculateion fare'
]
```

نام جدید ستون مسافت : Mean Distance

نام جدید ستون کرایه محاسباتی : Mean Calculateion fare

نرمالسازی بر روی دیتافریم path_stats انجام می دهیم. و با روش Elbow بهترین مقدار برای تعداد خوشها را بدست می آوریم. (کدهای Elbow این بخش همانند کد Elbow توضیح داده شده در بخش های قبل است) و نمودار آن را رسم می کنیم تا از روی نمودار زانویی را بیابیم.



همانطور که در نمودار روبه رو مشاهده می کنید ، مقدار زانویی را می توان ۳ یا ۴ درنظر گرفت. چون می خواهیم ببینیم کدام تعداد خوشه بهتر عمل می کند ، پس با هر دو مورد مدل KMeans را انجام می دهیم و در آخر مقایسه انجام می دهیم.

```
[ ] #KMeans Clustering
#k = 3
kmeans = KMeans(n_clusters = 3 , random_state=42)
path_stats['Cluster(k=3)'] = kmeans.fit_predict(scaled_features)
```

مقدار `n_clusters` را برابر ۳ قرار می دهیم زیرا می خواهیم تعداد خوشه ها ۳ باشد. سپس `kmeans.fit_predict` را که داده های نرمالسازی شده است را به `scaled_features` می دهیم تا خوشه بندی صورت گیرد و خروجی را در ستون جدید به نام `Cluster(k=3)` در دیتافریم `path_stats` ذخیره سازی می کنیم.

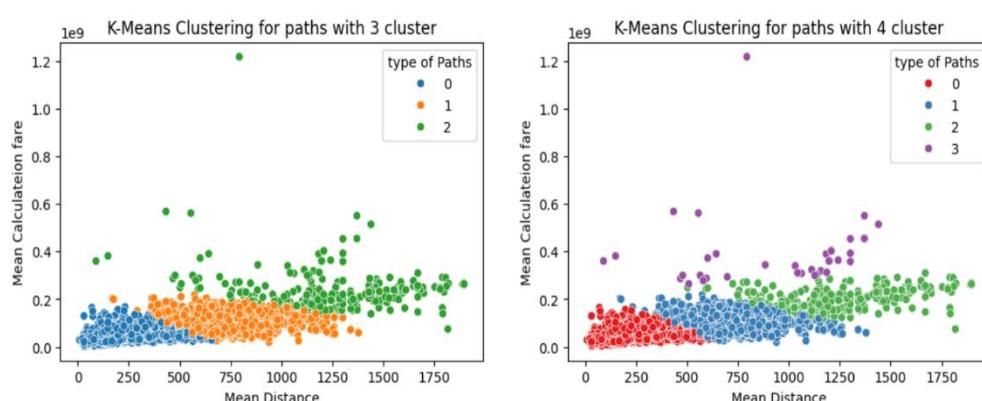
توضیحات کدهای گفته شده را برای خوشه بندی با تعداد خوشه ۴ هم انجام می دهیم تا مدل ساخته شود و خوشه بندی انجام شود. با این تفاوت که فقط مقدار `n_clusters` را ۴ قرار می دهیم.



```
#KMeans Clustering
#k = 4
kmeans = KMeans(n_clusters = 4 , random_state=42)
path_stats['Cluster(k=4)'] = kmeans.fit_predict(scaled_features)
```

حال زمان ان است که مصورسازی را براساس خوشه بندی انجام شده ، انجام دهیم. نمودار `scatterplot` از کتابخانه `seaborn` را برای مصورسازی انتخاب می کنیم. برای هر دو نمودار مقادیر `x = path_stats['Mean Distance']` ، `y = path_stats['Mean Calculateion fare']` از دیتافریم `path_stats` است. اما برای مقادیر `hue` برای نمودار اول ستون `(3)` قرار می دهیم زیرا می خواهیم تصویر خوشه بندی مسیرها را با ۳ خوشه مشاهده کنیم. و برای نمودار دوم مقدار `hue` را `(4)` قرار می دهیم تا تصویر خوشه بندی مسیرها را با ۴ خوشه مشاهده کنیم. از `(subplot)` استفاده کرده ایم تا دو نمودار را در یک سطر قرار دهد. دو نمودار درنهایت به

شکل زیر هستند :



دو نمودار زیر را
برای انتخاب خوشه
بندی مناسب
بررسی می کنیم .

نمودار اول : خوشه بندی با تعداد ۳ خوشه

در نمودار با تعداد خوشه ۳ تا تفکیک ساده تر شکل گرفته است. اما در برخی نقاط با کرایه های خیلی بالا (مثلًا نقاط بالا سمت چپ یا نقاط بالای نمودار) همه در یک خوشه افتاده اند . در واقع وقتی تعداد خوشه ها ۳ تاست ممکن است برخی از تفاوت های رفتاری یا هزینه ای مسیرها از بین برود؛

نموار دوم : خوشه بندی با تعداد ۴ خوشه

در خوشه بنفسش (خوشه ۳) به نظر می رسد بخشی از نقاط با کرایه های بالا و مسافت نسبتاً متوسط از سایر خوشه ها جدا کرده است؛ این یعنی توانسته الگوهای خاص با کرایه های بالا و غیرمعمول را کشف کند در در خوشه بندی با تعداد ۳ خوشه این امر حاصل نشده بود. توزیع خوشه ها متوازن تر به نظر می رسد و در نتیجه خوشه بندی با تعداد خوشه ۴ مناسب تر است.

پس ستون Cluster(k=3) را از دیتافریم حذف میکنیم. سپس براساس ستون Cluster(k=4) گروپ بای انجام می دهیم و براساس همان ستون میانگین گیری انجام می دهیم تا ویژگی های آماری را بدست بیاوریم و بتوانیم برچسب دهی به خوشه ها را انجام دهیم.

Cluster(k=4)	Mean Distance	Mean Calculation fare
0	250.300165	5.487439e+07
1	723.277652	1.038593e+08
2	1242.189091	1.988133e+08
3	931.055556	3.922716e+08

با توجه به این ویژگی های آماری بدست آمده می توانیم برچسب گذاری مربوط به مسیرها را انجام دهیم و بگوییم کدام خوشه ها کوتاه و کم کرایه ، کدام خوشه طولانی و با کرایه بالا و... هستند.

- خوشه ۰ : مسیر کوتاه و کم کرایه
- خوشه ۱ : مسیر نسبتاً کوتاه و نسبتاً کم کرایه

• خوشه ۲ : مسیر طولانی و نسبتا کم کرایه

• خوشه ۳ : مسیر نسبتا کوتاه و کرایه زیاد

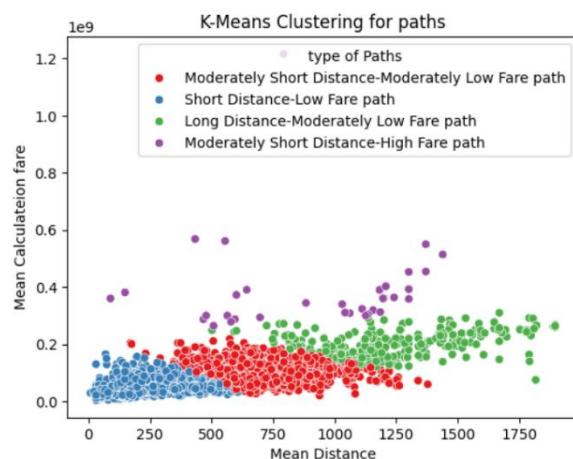
این برچسب ها را به انگلیسی مانند روال قبلی به دیتافریم می افزاییم که طبق کدهای زیر است.

```
▶ برجسب گذاری خوشه ها
path_label_map = {
    0 : 'Short Distance-Low Fare path',
    1 : 'Moderately Short Distance-Moderately Low Fare path',
    2 : 'Long Distance-Moderately Low Fare path',
    3 : 'Moderately Short Distance-High Fare path',
}

path_stats['path_label_map'] = path_stats['Cluster(k=4)'].map(path_label_map)
path_stats
```

path_col	Mean Distance	Mean Calculateion fare	Cluster(k=4)	path_label_map
اراک → تبریز	737.0	6.350000e+07	1	Moderately Short Distance-Moderately Low Fare ...
انبار نفت قم → آیگم محلات	118.0	2.000000e+07	0	Short Distance-Low Fare path
انبار نفت قم → آران و بیدگل	110.0	1.477083e+07	0	Short Distance-Low Fare path
انبار نفت قم → ابو زیداباد	137.0	1.700000e+07	0	Short Distance-Low Fare path
انبار نفت قم → اراک	134.0	6.000000e+07	0	Short Distance-Low Fare path
...
کلپیگان → قائم شهر	600.0	6.694932e+07	1	Moderately Short Distance-Moderately Low Fare ...
کلپیگان → مریوان	593.0	8.000000e+07	1	Moderately Short Distance-Moderately Low Fare ...
کلپیگان → گرگان	765.0	6.800000e+07	1	Moderately Short Distance-Moderately Low Fare ...
گمرک قم → تهران	148.0	1.032420e+08	0	Short Distance-Low Fare path
گمرک قم → شیراز (فارس)	748.0	2.179149e+08	2	Long Distance-Moderately Low Fare path

2290 rows × 4 columns



نمودار scatterplot را با قرار دادن y , x مانند رسم نمودار مسیرها براساس ستون خوشه مقدار دهی می کنیم. و مقدار hue را برابر ستون path_label_map می گذاریم که برچسبها در این ستون ذخیره شده اند. در قسمت راهنمای این برچسب ها هستندو نمودار به صورت زیر است :

KMeans Clustering خوشه بندی نوع بارها یا محصولات حمل شده با استفاده از روش

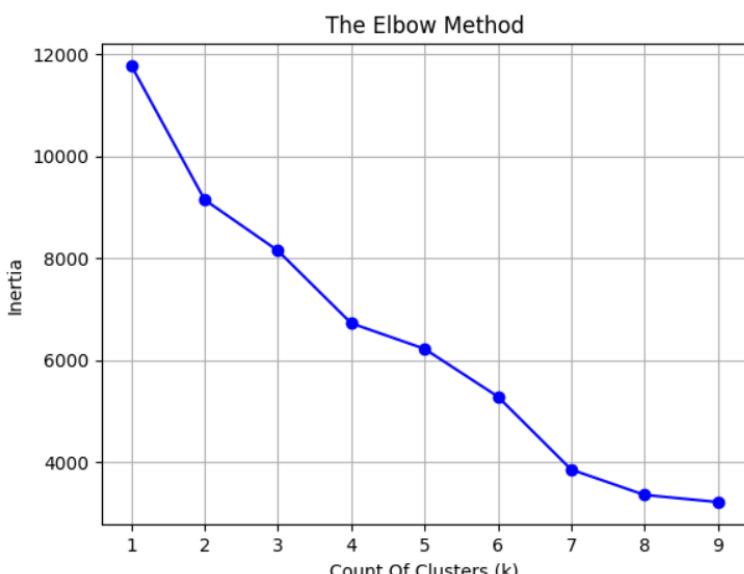
برای انجام این خوشه بندی ویژگی هایی که فکر میکنیم می توانند تاثیرگذار باشند را انتخاب می کنیم و در دیتافریم جدیدی به نام product_df ذخیره می کنیم. ویژگی هایی مانند Weight و Evacuation fee و Loading fee و Insurance amount و Calculateion fare و Distance و Product name . ستون نام محصولات (Value added insurance premium و rent) را هم در این دیتافریم ذخیره می کنیم.

براساس ستون نام محصول تجمعی انجام می دهیم و میانگین تمام ویژگی های دیگر را برای هر محصول محاسبه می کنیم. تا خوشه بندی را بر روی این دیتافریم که براساس نام محصول است، انجام دهیم. نامگذاری جدید برای ستون ها انجام می دهیم و ستون ایندکس، ستون نام محصول

نامگذاری جدید برای ستون های دیتابست #product_stats
product_stats.columns = ['Mean Weight', 'Mean Distance', 'MeanCalculateion fare', 'Mean Insurance amount', 'Mean Loading fee', 'Mean Evacuation fee', 'Mean rent', 'Mean Value added insurance premium']
product_stats

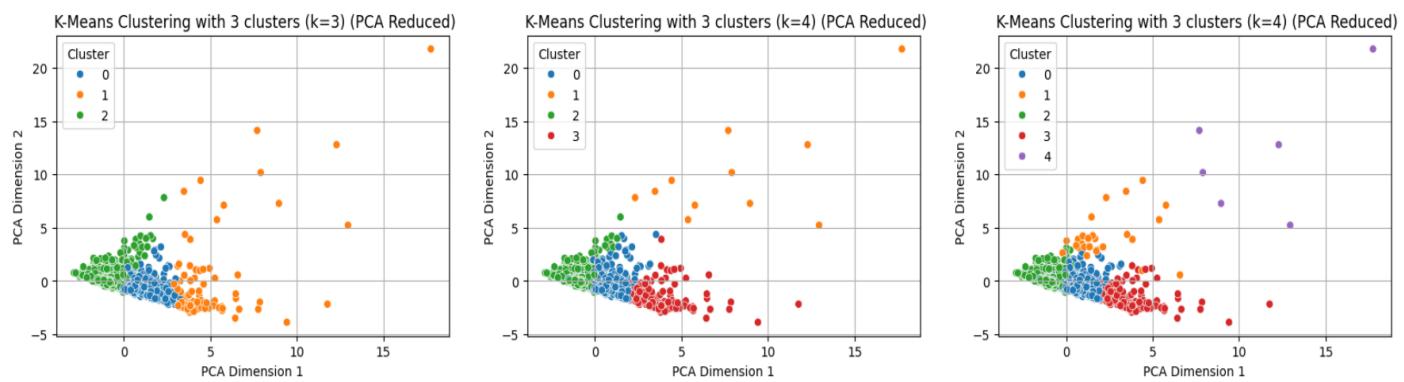
است.

(product_stats) نرمالسازی را بر روی این دیتافریم Elbow را مانند بخش های قبلی انجام می دهیم. روش Elbow را بهترین تعداد خوشه را بدست آوریم. نمودارش را هم رسم می کنیم که به صورت زیر است :



با توجه به نمودار بالا که از روش elbow آمده می توانیم ببینیم که زانویی این نمودار در نقطه ۳ است البته این زانویی تا ۴ و ۵ هم ادامه دارد. یعنی میتوان تعداد خوشه ها را از بین این سه عدد انتخاب کرد البته ما برای هر سه عدد تست می کنیم و هر کدام بهتر بود پس از تحلیل و

بررسی آن K مورد نظر را انتخاب میکنیم. مدل KMeans را برای $n_clusters$ با مقدار ۳، ۴ و ۵ انجام می دهیم. و ستون های بدست آمده از سه خوش بندی را به ترتیب (Cluster($k=3$) ، Cluster($k=4$) و Cluster($k=5$) نامگذاری می کنیم و در دیتا فریم ذخیره می کنیم. حالا نوبت به کاهش بعد به روش pca و رسم نمودار می رسد. مانند مدل سازی قبلی نمودارها را طبق همان زوال رسم می کنیم و درنهایت نمودارها به شکل زیر هستند :



با توجه به نمودارهای بالا میتوان اینگونه تحلیل کرد که بهترین تعداد خوش برای داده های مربوط به کالاهای ۴ تاست.

دلایل درستی انتخاب ۴ خوش به شرح زیر است :

- تعداد ۴ تا خوش باعث پوشش دهی بهتر داده های خاص شدند(داده های پرت یا گروه های متمایز).
- در نمودار ۱ که تعداد خوش ها ۳ تاست ، خوش نارجی بیش از حد بزرگ است و داده های متمایز در بالا و مرکز نمودار را با داده های معمولی قاطی کرده است.
- در نمودار ۲ که تعداد خوش ها ۴ تاست ، این داده های متمایز به درستی در خوش های جداگانه افتادند بدون اینکه خوش ها خیلی پراکنده شوند.
- در نمودار ۳ که تعداد خوش ها ۵ تاست ، تقسیم بندی بیش از حد شده و خوش های مشابه بی دلیل به دو قسمت شدند یعنی دو خوش مشابه به هم وجود دارد(خوش های ۱ و ۴).

- نمودار elbow هم نشان میداد که بعد از نقطه ۴ نرخ کاهش اینرسی کم می شود؛ یعنی ۴ تا خوشة نقطه زانو بوده است.

دو ستون مربوط به ۳ خوشة و ۵ خوشة را حذف می کنیم. برای برچسب دهی به خوشه ها مطابق روال همیشه، براساس خوشه Cluster($k=4$) (تجمع) میانگین می گیریم. و جدول زیر ویژگی های آمری خوشه ها را نشان می دهد:

	Mean Weight	Mean Distance	Mean Calculateion fare	Mean Insurance amount	Mean Loading fee	Mean Evacuation fee	Mean rent	Mean Value added insurance premium
Cluster($k=4$)								
0	13681.311381	548.345259	9.385121e+07	1.163179e+06	3.591988e+05	40602.325958	9.376098e+07	1.100177e+05
1	18577.732451	566.495323	1.644644e+08	2.394171e+07	2.057224e+06	955327.510917	1.641357e+08	2.365723e+06
2	5202.017601	462.248262	5.085147e+07	9.046282e+05	6.958775e+05	61211.512151	5.080321e+07	8.481970e+04
3	18477.725252	969.438944	1.910797e+08	1.929349e+06	3.397342e+05	89558.189372	1.910567e+08	1.799115e+05

نام خوشه ها براساس بررسی و تحلیل جدول بالا صورت زیر انتخاب شدند:

- خوشه ۰ : بارهای معمولی با مسیر و هزینه متوسط
- خوشه ۱ : بارهای سنگین و ارزشمند با هزینه های بالا و خدمات خاص
- خوشه ۲ : محموله های سبک برای مسافت های کوتاه با هزینه پایین
- خوشه ۳ : بارهای حجیم و سنگین در مسیرهای طولانی با کرایه بالا

برچسب ها را به انگلیسی تعریف می کنیم برحسب خوشه ها و آنها را با استفاده از تابع map() به هر خوشه تطبیق می دهیم و ستون برچسبها را با نام product_label_map تعریف می کنیم.

```
▶ ایجاد برچسب ها
product_label_map = {
    0 : 'Typical cargo with average cost' ,
    1 : 'High-value & insured heavy goods' ,
    2 : 'Lightweight short-range shipments' ,
    3 : 'Long-distance heavy cargo with high fare'
}

# اضافه کردن ستون برچسب ها
product_stats['product_label_map'] = product_stats[['Cluster(k=4)']].map(product_label_map)
product_stats
```

بارهای معمولی با مسیر و هزینه متوسط
بارهای سنگین و ارزشمند با هزینه های بالا و خدمات خاص
محموله های سبک برای مسافت های کوتاه با هزینه پایین
بارهای حجیم و سنگین در مسیرهای طولانی با کرایه بالا

چون ستون نام کالا به عنوان ستون ایندکس قرار گرفته کد زیر را میزنیم تا ستون ایندکس ریست شود و ایندکس شماره سطرها قرار گیرد.

در آخر هم نام محصولات را که در هر خوشه قرار دارند را خروجی می گیریم تا ببینیم هر خوشه شامل چه محصولاتی هستند.

بخش تشخیص ناهنجاری (Anomaly Detection)

تشخیص ناهنجاری برای پیدا کردن بارهایی با کرایه های بسیار بیشتر و یا کمتر از معمول در قدم اول برای تشخیص ناهنجاری بارهایی که کرایه هی خیلی زیاد و یا خیلی کم دارند ، باید ستون های مورد نیاز برای تشخیص ناهنجاری را جدا کنیم. برای این تشخیص ناهنجاری فقط به دو ستون Product name , Calculateion fare نیاز داریم ؛ این دو ستون را در دیتافریم جدیدی به نام product_fare_df ذخیره می نماییم.

برای تشخیص ناهنجاری از مدل Z-Score استفاده می نماییم. در ابتدا کتابخانه های مورد نیاز مانند from scipy.stats import zscore را ایمپورت می کنیم. سپس شروم به ساخت مدل می کنیم. Z-Score نشان می دهد هر داده چند انحراف معیار از میانگین فاصله دارد. فرمولش هم به صورت $Z = \frac{x - \bar{x}}{\sigma}$ است. اگر مقدار Z از ۳ یا -۳ بیشتر باشد، یعنی داده خیلی دور از میانگین است و به عنوان ناهنجار در نظر گرفته می شود. زیرا threshold (آستانه) را برابر ۳ قرار داده ایم ؛ یعنی داده هایی که بیشتر از ۳ انحراف معیار با میانگین فاصله دارند، ناهنجار هستند. و چون $threshold=3$ است ، یعنی حدود ۹۹.۷٪ داده ها ، نرمال تشخیص داده می شوند. با این خط هم outliers = np.where(np.abs(z_scores) > threshold) که اگر قدر مطلق مقدار z_scores بیشتر از ۳ باشد ، ناهنجار تشخیص دهد.

برای ستون کرایه محاسباتی ، میانگین و انحراف معیار را با دوتابع mean() ، std() محاسبه می نماییم. میانگین بدست آمده برابر با 73808590.53798878 و انحراف معیار برابر با

```
my_zscore = (product_fare_df['Calculateion fare'] - np.mean(product_fare_df['Calculateion fare'])) / np.std(product_fare_df['Calculateion fare'])
```

حال می خواهیم مقدار z-score را به صورت دستی محاسبه می کنیم تا ببینیم آیا مقادیر بدست آمده از Z-score از کتابخانه scipy با این مقداری که ما بدست آوریم یکسان است یا خیر.

سپس داده های پرت را خروجی می گیریم. درنهایت نمودار را رسم می کنیم.

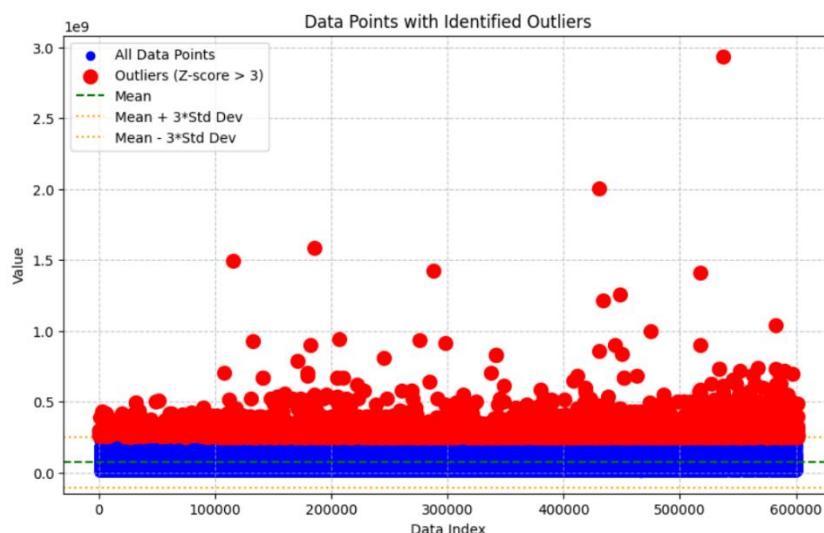
در خط اول کدهای بالا در ابتدا همه ی نقاط را رسم می کنیم. سپس در خط بعدی با استفاده از

```
▶ import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(range(len(product_fare_df['Calculateion fare'])), product_fare_df['Calculateion fare'], color='blue', label='All Data Points')
plt.scatter(outliers, np.array(product_fare_df['Calculateion fare'])[outliers], color='red', s=100, label='Outliers (Z-score > 3)', zorder=5) # zorder برای نمایش روی بقیه
plt.axhline(np.mean(product_fare_df['Calculateion fare']), color='green', linestyle='--', label='Mean')
plt.axhline(np.mean(product_fare_df['Calculateion fare']) + threshold * np.std(product_fare_df['calculateion fare']), color='orange', linestyle=':', label='Mean + 3*Std Dev')
plt.axhline(np.mean(product_fare_df['calculateion fare']) - threshold * np.std(product_fare_df['calculateion fare']), color='orange', linestyle=':', label='Mean - 3*Std Dev')

plt.title('Data Points with Identified Outliers')
plt.xlabel('Data Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

این کد [np.array(product_fare_df['Calculateion fare'])][outliers] نقاط پرت را در نمودار به رنگ قرمز رسم می کنیم. میانگین ستون کرایه محاسباتی را با خط سبز و این دستور $\text{np.mean}(\text{product_fare_df}[\text{'Calculateion fare'}]) + \text{threshold} * \text{np.std}(\text{product_fare_df}[\text{'Calculateion fare'}])$ و $\text{np.mean}(\text{product_fare_df}[\text{'Calculateion fare'}]) - \text{threshold} * \text{np.std}(\text{product_fare_df}[\text{'Calculateion fare'}])$



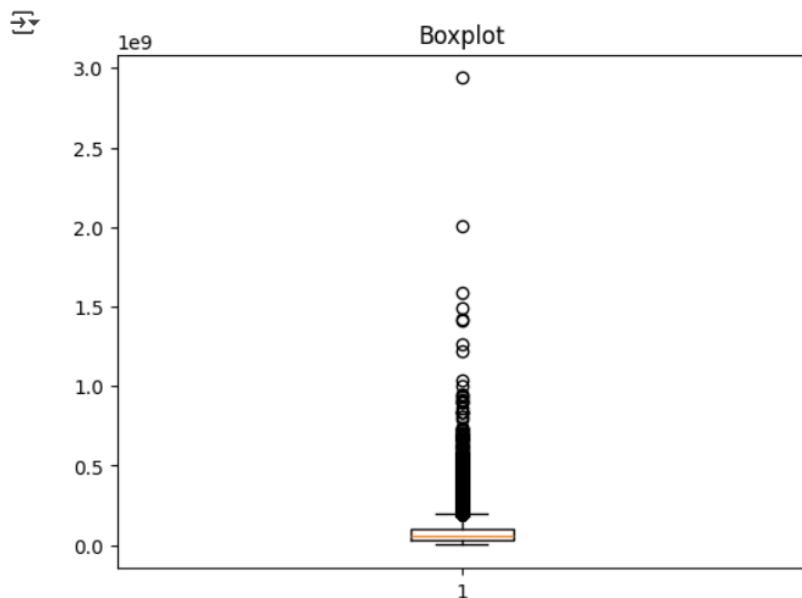
در دستور نمودار () که یک خط افقی که موازی محور x است رسم می کند. دو خط رسم می کند که مرز تشخیص ناهنجاری را نشان می دهد. خط بالایی (Mean + 3*Std Dev) مرز بالای ناحیه داده های نرمال

را نشان می دهد. خط پایینی (Mean - 3*Std Dev) را نشان می دهد.

مرز پایین ناحیه داده های نرمال را نشان می دهد. هر داده ای کوچکتر از و پایین تر از این خط باشد ، ناهنجار منفی شناسایی می شود. برای چه این خطوط را رسم کردیم ؟ رسم این خطوط ، کمک می کند به صورت بصری مشاهده کنیم ناحیه امن داده ها (نرمال) کجاست و در نتیجه هر نقطه خارج از این محدوده، به احتمال زیاد یک Outlier هست. و این خمطوط به رنگ نارنجی در نمودار مشخص شده اند.

با توجه به نمودار روبه رو ، تعداد زیادی از داده ها به عنوان ناهنجار علامت خورده اند (قرمز)، که نشان میدهد توزیع داده خیلی کشیده (long tail) هست. بعضی کرایه ها چندین برابر میانگین هستند (در حد میلیارد). این می تواند ناشی از ثبت اشتباه قیمت (Data Entry Error) ، بارهای خاص و خیلی سنگین یا مسافت بسیار طولانی و یا مقادیر قدیمی قبل از تغییر نرخ ها باشد.

```
▶ import matplotlib.pyplot as plt  
plt.boxplot(product_fare_df['Calculateion fare'])  
plt.title("Boxplot")  
plt.show()
```



نمودار جعبه ای برای ستون کرایه محاسباتی رسم می کنیم. با توجه به نمودار جعبه ای روبه رو هم تعداد داده های پرت زیاد است. یعنی مبلغ کرایه محاسباتی برای برخی از محصولات بسیار بالا بوده است. در حالی که داده های نرمال تا حدود ۱۰۰ تا ۲۰۰ میلیون بوده اند.

با استفاده از روش جنگل انزوا (Isolation Forest) هم تشخیص ناهنجاری برای کرایه محاسباتی انجام می دهیم.

الگوریتم جنگل ایزوله (Isolation Forest) یا جنگل جداسازی، یک الگوریتم یادگیری بدون نظارت برای تشخیص ناهنجاری (Anomaly) است که برای جداسازی نقاط پرت (Outlier) به کار می رود.

مدل جنگل انزوا را می سازیم و $\text{contamination}=0.05$ قرار می دهیم و این یعنی ۵٪ از داده ها را مدل ناهنجار شناسایی می کند. برای ستون کرایه محاسباتی (Calculateion fare) مدل را فیت می کنیم. سپس پیش بینی را با دستور زیر انجام می دهیم مانند کد زیر :

```
y_pred = isolation_forest.predict(product_fare_df[['Calculateion fare']])
```

خروجی این کد آرایه ای است که مقدارهای ۱ و -۱ دارند. ۱ نشان دهنده ای داده های هنجار و نرمال ، و مقدار -۱ نشان دهنده ای داده های ناهنجار هستند.y_pred را در ستون جدیدی به نام product_fare_df در دیتا فریم Anomaly_normal ذخیره می نماییم.

سپس با استفاده از کتابخانه plotly.express scatter() نمودار را برای رسم داده های هنجار و ناهنجار استفاده می کنیم. به این دلیل از این کتابخانه برای رسم نمودار استفاده می کنیم چون این

```
▶ import plotly.express as px

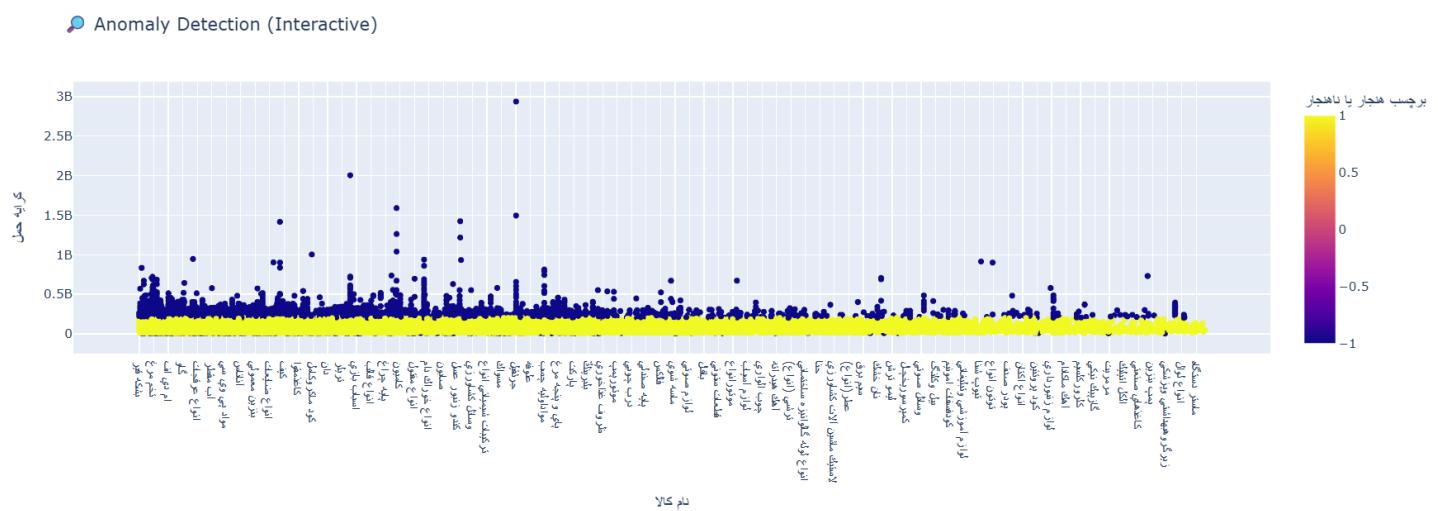
fig = px.scatter(
    product_fare_df,
    x=product_fare_df['Product name'],
    y='Calculateion fare',
    color='Anomaly_normal', #رنگ بر اساس عادی یا ناهنجار بودن
    hover_data={
        'Product name': True,
        'Calculateion fare': ':.2f',
        'Anomaly_normal': True
    },
    title='🔍 Anomaly Detection (Interactive)',
    labels={
        'Calculateion fare': 'کرایه حمل',
        'Product name': 'نام کالا',
        'Anomaly_normal': 'برچسب هنجار یا ناهنجار'
    }
)
#نمایش نمودار
fig.show()
```

کتابخانه قابلیت های خیلی بهتری نسبت به seaborn دارد ، همچنین از زبان فاسی هم پشتیبانی می کند و مصورسازی پیشرفته تری را نسبت به matplotlib کار بران قرار می دهد. همینطور می توانیم با نمودارهای این کتابخانه ، وقتی بر روی نقاط روی نمودار هاور

می کنیم اطلاعات مفیدی شامل هر آن چیزی که ما بگوییم نمایش می دهد (برای هر داده از دیتافریم موردنظر) و میتوان گفت به گونه ای نمودار تعاملی است.

در ابتدا دیتافریم موردنظر را در scatter() می گذاریم و سپس ستونهایی که میخواهیم در محورهای x , y باشند را مشخص می کنیم. نام محصول را روی محور x و کرایه محاسباتی را روی محور y قرار می دهیم. سپس مقدار color را ستون Anomaly_normal قرار می دهیم تا رنگبندی داده ها بر اساس این ستون باشد. سپس در قسمت hover_data (بخشی که مشخص می کنیم کدام داده ها در قسمت هاور نمایش داده شوند)، ستون های نام محصول، کرایه محاسباتی و ستون قارا می دهیم و نام هایی که قرار است برای هر ستون هم در قسمت هاور نمایش داده شود هم مشخص می کنیم. در آخر نموداری مانند زیر داریم:

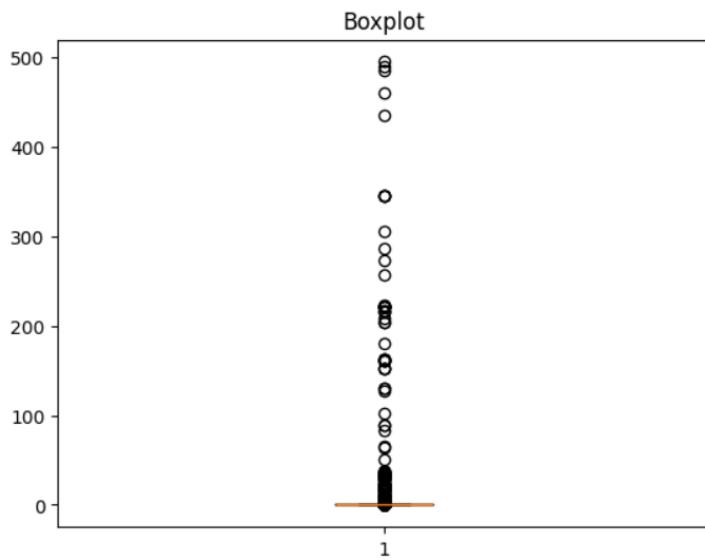
همانطور که در قسمت راهنمای دیده می شود مقادیر ۱ با رنگ زرد و مقادیر -۱ با رنگ آبی نمایش داده می شوند.



همانطور که در نمودار بالا دیده می شود این نمودار بسیار با کیفیت تر و زیباتر داده ها را نمایش می دهد. و داده های پرت برای کرایه های بالا، بیشتر از داده های پرت برای کرایه های پایین است.

تشخیص ناهمجارتی و کشف زمان های غیرمعمول در تحويل کالا

اگر به یاد داشته باشید در بخش‌های قبل (رگرسیون خطی) ستون مدت زمان رسیدن کالا به دست مشتری را بدست آوریم و Delivery_days نامیدیم. می‌خواهیم تشخیص ناهنجاری را برای این ستون انجام دهیم؛ در ابتدا نمودار جعبه‌ای (boxplot) را برای این ستون رسم می‌نماییم.

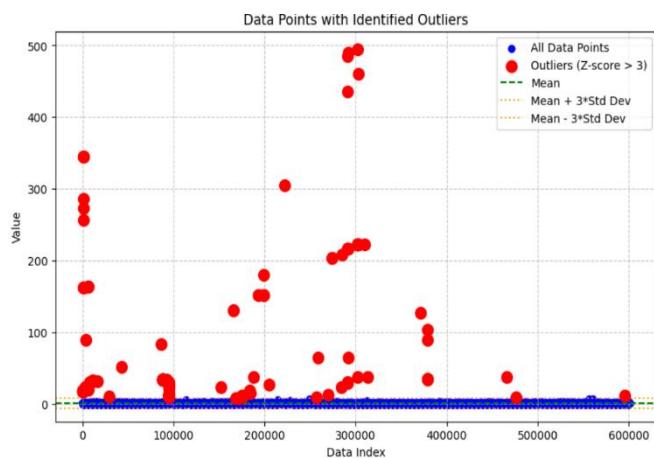


با توجه به نمودار جعبه‌ای روبه رو، داده‌های پرت دیده می‌شوند. داده‌ها در چارک اول، میانه و چارک سوم (دامنه میان چارکی) از ۲۰ هم کمتر است. یعنی اکثر داده‌ها در ستون مدت زمان رسیدن کالا به دست مشتری از ۱ تا ۲۰ روز بوده است. ماکسیمم مقدار برای داده‌های پرت تقریباً ۵۰۰ روز بوده است؛ به آن

معنا که کالایی وجود داشته است که ۵۰۰ روز طول کشیده است تا به دست مشتری (گیرنده) برسد.

برای اینکه بتوانیم دید بهتری نسبت به داده‌های پرت برای این ستون (Delivery_days) داشته باشیم روش Z-Score پیاده‌سازی می‌نماییم.

همانند تشخیص ناهنجاری به روش Z-Score برای کرایه محاسباتی، طبق همان روال برای این



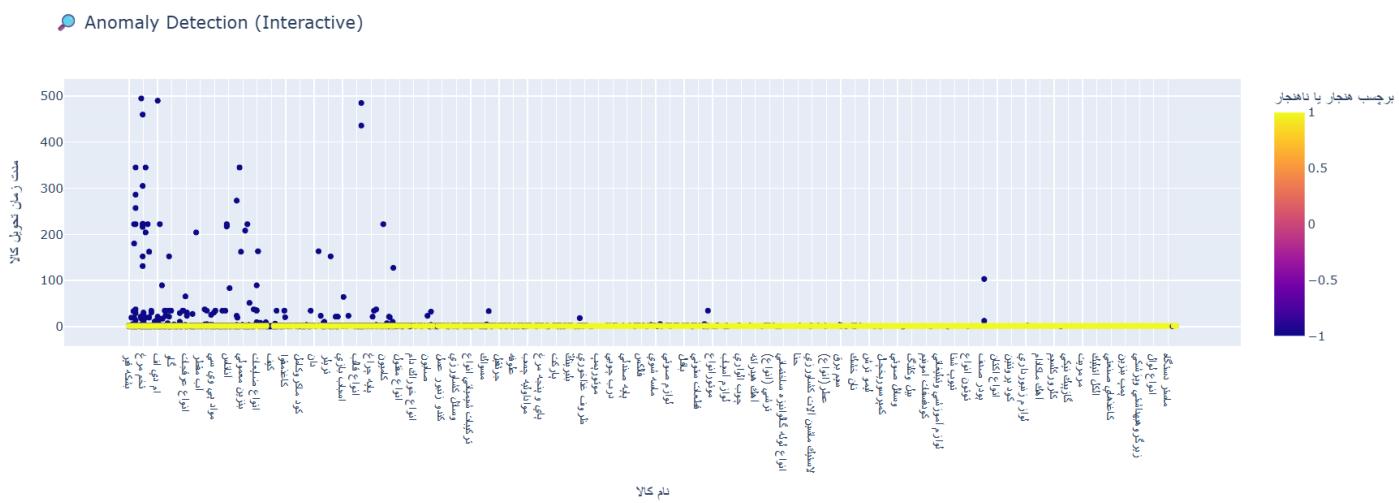
ستون (Delivery_days) هم انجام می‌دهیم (توضیحات مشابه با تشخیص ناهنجاری به روش Z-Score برای کرایه محاسباتی). تنها تفاوت این است که ستونی که باید به مدل Z-Score دهیم Delivery_days، سپس نمودارش را رسم می‌کنیم که به صورت زیر خواهد بود:

داده هایی که به رنگ قرمز هستند ، داده های پرت هستند که مقدارشان منطقی به نظر می رسد و داده های آبی رنگ هم ، هنجار هستند.

روش جنگل انزوا (Isolation Forest) هم برای تشخیص ناهنجاری برای ستون Delivery_days انجام می دهیم

همانند روندی که برای تشخیص ناهنجاری برای کرایه محاسباتی به روش جنگل انزوا انجام دادیم را ، انجام می دهیم. تنها تفاوت این است که ویژگی ای برای تشخیص ناهنجاری به مدل می دهیم تغیر کرده است. نمودار تشخیص ناهنجاری مدت زمان رسیدن کالا به دست مشتری به صورت زیر

است :



همانطور که مشخص است نقاط آبی داده های پرت هستند که توسط مدل جنگل انزوا مشخص شده اند. نقاط زرد هم داده های نرمال دیتاست هستند.

تشخیص ناهنجاری و کشف رکوردهایی با وزن بالا اما هزینه پایین و یا بالعکس

در تشخیص ناهنجاری برای این مسئله ی مطرح شده باید از دو ستون Weight و fare استفاده کنیم. قبل از اینکه این دو ویژگی را به مدل دهیم ، ویژگی ها را نرمال می کنیم و داده

```
#1 و 2 برچسب زدن به
transport_df['Anomaly_weight_fare_label'] = transport_df['Anomaly_weight_fare'].map({1:'normal' , -1:'anomaly'})
```

ی نرمالسازی شده را به مدل جنگل انزوا می دهیم. چون همانطور که مشخص است تفاوت زیادی بین اعداد دو ستون وزن و کرایه محاسباتی وجود دارد. طبق روال همیشگی مدل را می سازیم و نمودار داده های هنجار و ناهنجار را رسم می کنیم. فقط یک نکته ای که وجود دارد این است که ما ستونی جدید از برچسب های **normal** , **anomaly** ایجاد کردیم که بر حسب همان ستون **Anomaly_weight_fare** است که داده های نرمال و ناهنجار با ۱ و -۱ مشخص شده اند.

نمودار به صورت زیر است :



تحلیل نمودار :

در ناحیه آبی که نشان دهنده داده های نرمال و هنجار هستند، رابطه ای منطقی بین وزن و کرایه دیده می شود. عموماً وقتی وزن افزایش پیدا می کند، کرایه هم در یک محدوده قابل پیش بینی افزایش پیدا می کند. بخش مرکز آبی تقریباً بین وزن ۰ تا ۲۵ کیلو و کرایه نزدیک به صفر یا چند صد میلیون قرار دارد. نقاط قرمز که نشان دهنده داده های ناهنجار هستند، نظر مدل، خارج از الگوی کلی هستند. دو نوع الگوی ناهنجار اینجا دیده می شود:

وزن بالا و کرایه پایین : یعنی حمل بار سنگین با هزینه بسیار کم (ممکنه داده اشتباه باشه یا قیمت گذاری غیرعادی) .

وزن پایین و کرایه بالا : یعنی باری سبک است اما هزینه اش غیرمعمول بالاست (ممکن است شامل

خدمات ویژه باشد و یا اشتباه ثبت داده باشد).

مخصوصاً بعد از وزن حدود k_{25} به بالا، بیشتر نقاط قرمز هستند چون کرایه‌ها در این محدوده رفتار یکنواختی ندارند و بعضاً خیلی پایین یا خیلی بالا می‌شوند. از محدوده k_{30} تا k_{60} وزن کالا، تقریباً همه نقاط قرمز هستند. این نشان می‌دهد که مدل رابطه منطقی بین وزن و هزینه را در این بازه پیدا نکرده، احتمالاً چون داده‌ها کم یا خیلی متنوع بودند.

بخش تحلیل زمانی

روند تغییر قیمت کرایه یا بیمه در سال‌ها و ماه‌ها؛ بیشترین بارگیری در چه ماه‌هایی انجام می‌شود؟

برای حل این تحلیل باید ستون‌های تاریخی را که دیتا تایپشان آبجکت است را به تاریخ تبدیل کنیم و ما قبل این کار را کرده‌ایم. حال با توجه به عنوان سوال که گفته شده «روند تغییر قیمت یا بیمه در سال‌ها و ماه‌ها» باید ستونی ایجاد کنیم که فقط شماره‌ی ما‌ها در آن ذخیره شده باشند. چون در دیتابس فقط مربوط به سال ۱۴۰۳ است پس ستونی مربوط به سال ایجاد نمی‌کنیم. این کار را به کمک تابع `dt.to_period` انجام می‌دهیم و مقدار 'M' را که مشخصه `month` است را در تابع مذکور قرار می‌دهیم. البته این کار را بر روی ستون تاریخ ارسال کالا (`Date`) انجام می‌دهیم یعنی شماره‌ی ما‌این ستون را در ستون دیگر ذخیره می‌کنیم.

```
#ایجاد ستون ماه از ستون Issue month
transport_df['Issue month'] = transport_df['Date of issue'].dt.to_period('M')
```

نام ستون جدید را `Issue month` نامگذاری می‌کنیم.

حال چون در صورت مسئله ذکر شده «روند تغییر قیمت کرایه یا بیمه» ما هر دو ویژگی را در نظر می‌گیریم. چون میخواهیم روند تغییر این دو ویژگی را در ماه‌های مختلف بررسی کنیم باید براساس ستون جدیدی که ایجاد کردیم (`Issue month`) تجمعی انجام دهیم و برای هر ماه میانگین کرایه

و بیمه را بدست بیاوریم؛ براساس این توضیحات ، ستونی که گروپ بای براساس آن صورت می گیرد ، به عنوان ایندکس آن دیتافریم ایجاد شده قرار میگیرد ؛ برای جلوگیری از این مسئله دستور reset_index() را هم در انتهای این خط کد می آوریم تا شماره سطرها به عنوان ایندکس دیتافریم قرار گیرد. نام دیتافریم را monthly_costs می گذاریم.

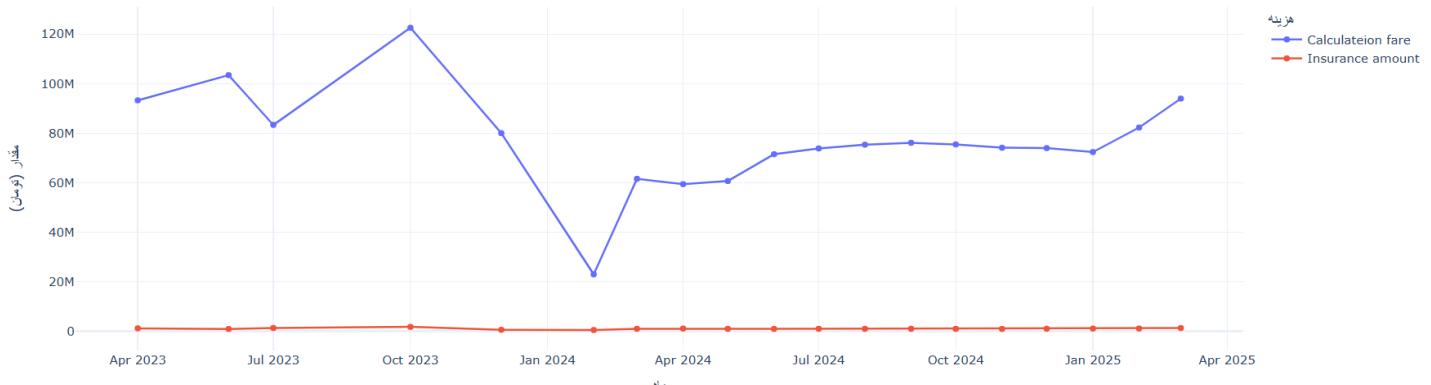
```
# محاسبه میانگین کرایه و بیمه در هر ماه
monthly_costs = transport_df.groupby('Issue month')[['Calculation fare', 'Insurance amount']].mean().reset_index()
```

برای سازگاری با نمودارهایی که DatetimeIndex نیاز دارند ، و فرمت خواناتر تاریخ در خروجی، بر روی ستون Issue month اجرا می کنیم. همینطور دلیل اصلی انجام این کار این است که این دستور در واقع داره ستون Issue month را که نوع داده‌ی Period هست، را به یک datetime64[ns] Timestamp تبدیل می کند و چون روز در این ستون مشخص نیست برای هیچ کدام از تاریخ‌های ذخیره شده ، پیش‌فرض را اول ماه در نظر می‌گیرد. و برای همه سلول‌ها و تاریخ‌ها این ستون اولین روز از آن ماه را قرار می‌دهد. پس این تابع برای داده‌های فقط ماه-سال، روز را به طور پیش‌فرض ۱ قرار میدهد.

```
monthly_costs['Issue month'] = monthly_costs['Issue month'].dt.to_timestamp()
```

برای رسم نمودار برای هر دو ویژگی کرایه محاسباتی و مبلغ بیمه ، از نمودار line() در کتابخانه plotly.express استفاده می کنیم. دیتافریم موردنظر (monthly_costs) و محور x را ستون Issue month می‌گیریم که این دستور در واقع داره ستون Issue month را که نوع داده‌ی Period هست، را به یک datetime64[ns] Timestamp تبدیل می کند و چون روز در این ستون مشخص نیست برای هیچ کدام از تاریخ‌های ذخیره شده ، پیش‌فرض را اول ماه در نظر می‌گیرد. و برای همه سلول‌ها و تاریخ‌ها این ستون اولین روز از آن ماه را قرار می‌دهد. پس این تابع برای داده‌های فقط ماه-سال، روز را به طور پیش‌فرض ۱ قرار میدهد.

روند میانگین کرایه و بیمه به تفکیک ماه



و برای محور `u` ، دو ستون کرایه محاسباتی و مبلغ بیمه دیتا فریم `monthly_costs` قرار می دهیم. برای نمایش نقاط بر روی خطوط در نمودار گزینه `markers=True` قرار می دهیم. در `Plotly` ، پارامتر `'x unified'` تعیین می کند که وقتی ماوس روی نمودار حرکت می کند، اطلاعات تمام خطوطی که روی یک محور `X` قرار دارند به صورت یکجا و یک پنجره مشترک نمایش داده شود. در نهایت نمودار به صورت زیر خواهد بود :

همانطور که مشاهده می کنید ، خط قرمز که مربوط به روند تغییر مبلغ بیمه را در ماه های مختلف نشان می دهد و روند تغییرش آنچنان تغییر بزرگی در تغییر قیمت به چشم نمی خورد و می توان گفت بیشترین و بالاترین مبلغ بیمه برای ماه Oct 2023 که ۱.۸۰ ۲۷۵ میلیون است. اما در صورتی که خط آبی که مربوط به روند تغییر کرایه محاسباتی است ، نوسانات زیادی داشته و در Oct 2023 در بالاترین حد خود بوده است یعنی کرایه محاسباتی ۱۲۲.۵۵ میلیون بوده است ؛ همچنین کمترین مقدار کرایه محاسباتی در Feb 2024 مقدار ۲۳ میلیون را داشته است.

تحلیل فصلی برای حمل بعضی محصولات خاص (در چه فصلی حمل بعضی محصولات بیشتر است؟)

چون که در صورت سوال آمده است «تحلیل فصلی» پس باید از ستون تاریخ ارسال (`Date of issue`) فصول سال را استخراج کنیم. تابعی تعریف می کنیم به نام `season` و به کمک

که در کتابخانه `datetime` است، فقط شماره ماه آن تاریخ مدنظر را می گیریم و در متغیری مانند `month` ذخیره میکنیم و شرط می گذاریم که اگر `1` ، `2` و یا `3` بود نام فصل را زمستان برگرداند؛ اگر `4` ،

```
تعريف تابع استخراج فصل از تاریخ
def season(date):
    month = date.month
    if month in [1,2,3]:
        return 'Winter'
    elif month in [4,5,6]:
        return 'Spring'
    elif month in [7,8,9]:
        return 'Summer'
    else :
        return 'Autumn'

#عمل تابع فصل بر روی ستون مورد نظر#
transport_df['season'] = transport_df['Date of issue'].apply(season)

transport_df.head()
```

۵ و یا ۶ بود نام فصل را بهار، اگر ۷، ۸ و یا ۹ بود تابستان و در غیر این صورت فصل را پاییز برگرداند طبق کد زیر:

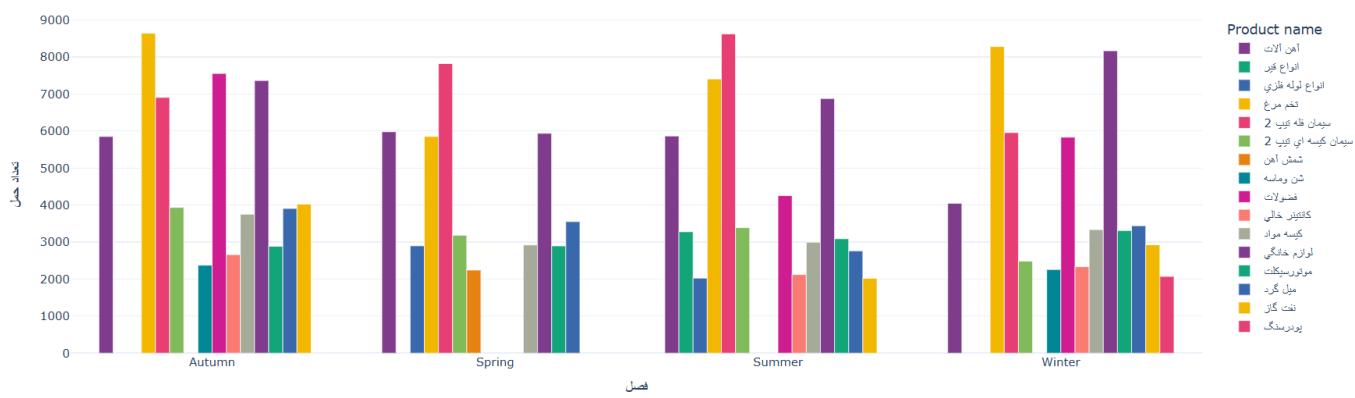
و این تابع season را بر روی ستون تاریخ ارسال (Date of issue) اعمال می کنیم و ستونی جدید به نام season در دیتاست اصلی ایجاد می کنیم.

بر اساس دو ستون فصل و نام محصول تجمعی و با استفاده از تابع size() تعداد حمل برای هر محصول را برای هر فصل محاسبه می کنیم. سپس برای اینکه شماره سطرها به عنوان ایندکس قرار index با Series size() groupby و با index_reset() یک چندسطحی است (Product name و season). این ها را به ستون های index reset_index() .

معمولی بر می گرداند. پارامتر name='Shipment count' اسم ستونی که شمارش ها داخلش می آید را Shipment count قرار می دهد. در نهایت اسم دیتافریم جدید را که حاصل این تجمعی است را seasonal_product_counts می گذاریم.

می خواهیم محصولاتی را بر روی نمودار بیاوریم که تعداد حملشان (Shipment count) بیشتر از ۲۰۰۰ باشد. سپس نمودار bar از کتابخانه plotly.express استفاده می کنیم. برای محور X ستون فصل و برای محور Y ستون Shipment count (تعداد حمل) را قرار می دهیم و ستون کالا را به پارامتر color می دهیم تا رنگبندی براساس کالاهای انجام پذیرد.

حمل محصولات در فصل های مختلف



اگر بر روی هر ستون که معرف یک محصول است موس را حرکت دهیم ، اطلاعات مربوط به آن محصول فصل ، تعداد حمل و نام کالا نمایش داده خواهد شد. بیشترین تعداد حمل برای کالای تخم مرغ است که ۸۶۴۳ بار حمل شده است و در فصل پاییز هم بوده است. در فصل بهار بالاترین تعداد حمل متعلق به سیمان فله تیپ ۲ با تعداد حمل ۷۸۲۲ بوده است. در فصل تابستان کالای سیمان فله تیپ ۲ با تعداد حمل ۸۶۲۴ است. این محصول در دو فصل به طور متدائل بیشتر تعداد حمل را داشته است پس کالایی است که مشتریان زیاد سفارش داده اند. در فصل زمستان کالای تخم مرغ با تعداد حمل ۸۲۸۳ است؛ این کالا هم در دو فصل به طور متدائل بیشترین تعداد حمل را داشته است.

بخش تحلیل مکانی

نقشه مسیرهای پرتردد بسازید

برای اینکه بتوانیم نقشه مسیرهای پرتردد را بسازیم و رسم کنیم نیاز به دو ستون مهم در دیتابیست The (Name of the city of origin) و نام شهر مقصد (name of the destination city) است، یعنی ستون نام شهر مبدا (transport_df) نیاز داریم. پس برای اینکه بتوانیم متوجه شویم هر مسیر (مبدأ-مقصد) چند بار تکرار شده است (یعنی برای هر مسیر شمارش انجام می دهیم) کد زیر را انجام می دهیم (تجمیع براساس این دو ستون مذکور انجام می دهیم) و برای جلوگیری از اینکه دو ستون مبدا و مقصد به عنوان ستون ایندکس قرار گیرند، از rest_index استفاده و نام ستون جدید را Rout count در پارامتر name می گذاریم.

# شمارش تعداد مسیرها (شهر مبدا و مقصد)	route_counts = transport_df.groupby(['Name of the city of origin' , 'The name of the destination city']).size().reset_index(name='Rout count')
--	--

حال که تعداد هر مسیر بدست آمد ، نام شهرهای مبدا را از دیتابریم route_count ، بدون تکرار در یک دیتابریم به نام unique_origin_cities و نام شهرهای مقصد را هم بدون تکرار و مقادیر یونیک را در دیتابریمی به نام unique_destination_cities ذخیره می کنیم. برای این کار از تابع

استفاده می کنیم. و برای اینکه بدانیم تعداد یونیک شهرهای مبدا و مقصد چقدر هستند با تابع `nunique()` استفاده می کنیم و این تابع تعدادشان را بر می گرداند. ۳۵ شهر مبدا بدن تکرار و ۱۰۲۹ شهر مقصد بدون تکرار داریم.

برای اینکه می خواهیم موقعیت جغرافیایی (طول و عرض جغرافیایی) را برای هر دو ستون شهر مبدا و شهر مقصد که اکنون دو دیتا فریم برای این دو ستون داریم که نام شهرها بدون تکرار در آنها ذخیره شده اند (`unique_destination_cities` و `unique_origin_cities`)، بدست آوریم، بهتر است اجتماع این دو دیتا فریم را در یک دیتا فریم ذخیره کنیم تا تابعی که برای بدست آوردن موقعیت جغرافیایی این شهرها می خواهیم استفاده کنیم و اعمال کنیم ، فقط یکبار باشد. این دیتا فریم اسمش `all_unique_cities` است.

```
▶ # دو لیست بالا را یکی می کنیم
# بدون تکرار لیست میکند نام شهرها را
all_unique_cities = pd.Series(list(set(unique_origin_cities) | set(unique_destination_cities)))
all_unique_cities
```

در کل ۱۰۳۸ نام شهر به صورت یونیک داریم برای دیتاست اصلی یعنی `transport_df` طبق کد زیر که تصویرش را می بینید ، می توانیم مختصات شهرها را استخراج کنیم. عرض جغرافیایی است. `sleep` برای ایجاد تأخیر بین درخواست هاست تا API ما را بلاک نکند. `tqdm` نوار پیشرفت هنگام اجرای لوب را نمایش می دهد.

```
import pandas as pd
from geopy.geocoders import Nominatim
from time import sleep
from tqdm import tqdm
```

حال روند کار کرد تابع گرفتن مختصات با مدیریت خطای get_coords را توضیح می دهیم ؛
تابع get_coords با یک نام شهر ورودی می گیرد. geolocator.geocode تلاش می کند موقعیت را از OSM (OpenStreetMap) مخفف OSM) می شود. یک پروژه متن باز و رایگان که یک «نقشه جهان» رو می سازه و همه می توانند در آن مشارکت کنند) بگیرد. timeout=10 محدودیت زمانی است. اگر نتیجه باشد، یک pd.Series([lat, lon]) بر می گردد؛ در غیر این صورت (None, None) تا

```
تابع گرفتن مختصات با مدیریت خطای
def get_coords(city_name):
    try:
        location = geolocator.geocode(f'{city_name}, Iran', timeout=10)
        if location:
            return pd.Series([location.latitude, location.longitude])
        else:
            return pd.Series([None, None])
    except Exception as e:
        return pd.Series([None, None])
```

بدانیم موقعیت جغرافیایی آن شهر پیدا نشده است. خطای احتمالی هم گرفته می شود و بر می گردد (None, None) برای برنامه قطع نشود.

برای بررسی اینکه اگر قبلًا مختصاتی ذخیره شده، از همون استفاده کند ، لازم است به این صورت کدها را سازماندهی کنیم ؛ اگر فایل iran_city_coords.csv از قبل وجود داشته باشد (چون ممکن است اجرای قبلی نیمه کاره قطع شده باشد)، آن را می خوانیم تا ادامه کار از همانجا انجام شود. set done_cities مجموعه شهرهایی است که قبلًا مختصات شان ذخیره شده است ؛ استفاده از

```
بررسی اینکه اگر قبلًا مختصاتی ذخیره شده، از همون استفاده کند
try:
    coords_df = pd.read_csv("iran_city_coords.csv")
    done_cities = set(coords_df['city'])
except:
    coords_df = pd.DataFrame(columns=["city", "lat", "lon"])
    done_cities = set()
```

برای جستجوی سریع است. اگر DataFrame نیست، یک done_cities می سازیم و خالی می شود.

برای فیلتر کردن شهرهایی که هنوز مختصات جغرافیایی ندارند اینگونه عمل می کنیم ؛ done_cities ~all_unique_cities.isin(done_cities) یعنی فقط آن شهرهایی را نگه دار که در نیستند. بنابراین فقط روی شهرهای باقی مانده کار می کنیم.

```
# فیلتر شهرهایی که هنوز مختصات ندارند
pending_cities = all_unique_cities[~all_unique_cities.isin(done_cities)]
```

برای قسمت دریافت مختصات برای شهرهای باقی مانده ، اینگونه عمل می کنیم ؛ لوب روی pending_cities اجرا می شود و برای هر شهر get_coords فراخوانی می شود. نتیجه را به اضافه می کنیم (اینجا از DataFrame تک-ردیفه استفاده شده). sleep(2.5) بین هر درخواست باعث می شود فاصله درخواستها حداقل ۲.۵ ثانیه شود. این کار برای

```
# دریافت مختصات برای شهرهای باقیمانده
for city in tqdm(pending_cities, desc="در حال دریافت مختصات"):
    lat, lon = get_coords(city)
    coords_df = pd.concat([coords_df, pd.DataFrame([
        {"city": city,
         "lat": lat,
         "lon": lon
    }])], ignore_index=True)

# تأخیر بین درخواستها برای جلوگیری از بلاک شدن
sleep(2.5)

# ذخیره دوره ای برای اینکه اگه قطع شد ادامه بدیم
if len(coords_df) % 20 == 0:
    coords_df.to_csv("iran_city_coords.csv", index=False)

# ذخیره نهایی
coords_df.to_csv("iran_city_coords.csv", index=False)
print("مختصات همه شهرها ذخیره شد")
```

```
0
city      0
lat       77
lon       77
dtype: int64
```

جلوگیری از Rate limit مفید است. هر ۲۰ ردیف، فایل به صورت دوره ای ذخیره می شود تا در صورت قطع، از همانجا ادامه بدهیم. و در نهایت ، پس از اتمام حلقه، فایل نهایی ذخیره می شود و پیام تأیید چاپ می شود.

سپس فایل CSV خروجی تابع گرفتن مختصات را دانلود می کنیم و دوباره

همین فایل را بارگذاری می کنیم. و از این فایل ، از اینجا به بعد استفاده خواهیم کرد. نام این فایلی که دانلود کردیم city_locations می گذاریم . با دستور isnull().sum() می بینیم آیا دیتافریم city_locations مقادیر خالی وجود دارند یا خیر و در خروجی ۷۷ تا طول و عرض جغرافیایی مقادیر خالی دارند.

برای اینکه این مقادیر خالی از بین روند ، این سطرها را حذف می نماییم . می توانیم به صورت دستی مطول و عرض جغرافیایی این شهرهایی که ستون طول و عرضشان خالیست را پیدا کنیم اما من ترجیح دادم حذفشان کنم. با دستور زیر این کار را می نماییم :

```
[26] city_locations_cleaned = city_locations.dropna(subset=['lat', 'lon'])
city_locations_cleaned.isnull().sum()
```

خروجی هم می گیریم تا ببینیم بعد از این دستور دوباره مقادیر خالی داریم یا تا
 اطمینان حاصل کنیم و دیده می شود که مقادیر خالی وجود ندارند.

	0
city	0
lat	0
lon	0

 dtype: int64

حال می خواهیم دو دیتافریم city_location و route_counts را به هم متصل کنیم
 (merge) تا به جدول مسیرها، مختصات مبدأ (longitude و latitude) را اضافه کنیم.
 ستون left_on='Name of the city of origin' که شهر مبدأ رو
 مشخص می کند. ستون right_on='city' (در دیتافریم city_locations_cleaned)
 که اسم شهر رو دارد. how='left' یعنی همه ردیفهای جدول سمت چپ (route_counts) نگه
 داشته شود و اگر برای شهری مختصات پیدا شد، به دیتافریم اضافه شود؛ اگه پیدا نشد، NaN در
 دیتافریم قرار گیرد. نام ستون های lat و lon را هم عرض جغرافیایی مبدا و طول جغرافیایی مبدأ
 (Origin_lat , Origin_lon) تغییر می دهیم.

 # merge مختصات مبدأ

 route_counts = route_counts.merge(city_locations_cleaned, left_on='Name of the city of origin', right_on='city', how='left')

 route_counts.rename(columns={'lat': 'Origin_lat', 'lon': 'Origin_lon'}, inplace=True)

برای اینکه مختصات برای شهرهای مقصد هم اضافه شوند ، مانند بالا براساس ستون نام شهر مبدأ
 که در پارامتر left_on قرار می دهیم ، اتصال را انجام می دهیم. نام ستون های lat و lon را به
 destination_lon و destination_lat تغییر می دهیم.

 انجام می دهیم merge حالا براساس شهر مقصد#

 route_counts = route_counts.merge(city_locations_cleaned, left_on='The name of the destination city', right_on='city', how='left')

 route_counts.rename(columns={'lat': 'destination_lat', 'lon': 'destination_lon'}, inplace=True)

سپس دو ستون city_x , city_y که حاصل از دو city_location merge به دیتافریم اضافه شدند
 را با دستور route_counts.drop(columns=['city_x' , 'city_y'], inplace=True) حذف می
 کنیم. بعد از این عملیات ها ، برای اطمینان از اینکه در دیتافریم مقادیر خالی وجود دارند یا خیر

، زیرا انتظار می رود مقادیر خالی وجود داشته باشند چرا که ۷۷ تا از شهرها را نتوانستیم موقعیت جغرافیاییشان را بدست بیاوریم و آنها را از دیتافریم city_locations حذف کردیم ولی در این دیتافریم آن شهرها وجود دارند و ممکن است چند باری به عنوان شهر مبدا و مقصد تکرار شده باشند زیرا در این دیتافریم ما تعداد تکرار مسیرها را شمرده ایم. پس دستور Origin_lon و Origin_lat و route_counts.isnull().sum() می زنیم و ۴ تا مقدار خالی در دو ستون destination_lon و destination_lat وجود دارند؛ این سطرها را هم حذف می کنیم و نام دیتافریم تمیزشده را route_counts_cleaned می گذاریم. سپس این دیتافریم را به صورت نزولی مرتب سازی می کنیم زیرا می خواهیم مسیرهای پرتردد را استخراج نماییم.

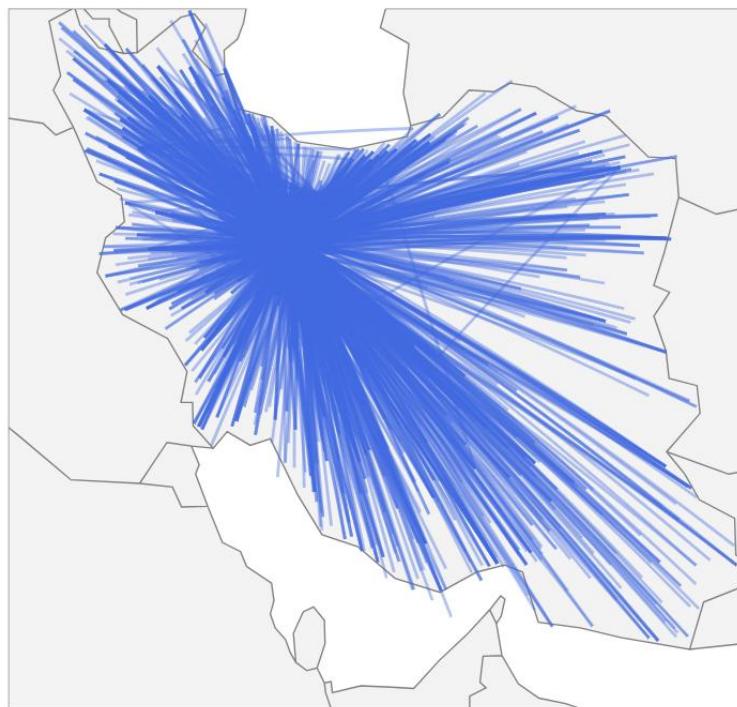
برای اینکه بتوانیم نمودار نقشه ای رسم کنیم نیاز است این کتابخانه را ایمپورت کنیم import Plotly. این دستور fig = go.Figure() . plotly.graph_objects as go می سازه که بعداً ترسیمات (Trace) رو بهش اضافه می کنیم.

حلقه ای روی دادهها برای ایجاد خطوط مسیر ایجاد می کنیم. این تابع ()iterrows() دادهها رو سطر به سطر از دیتافریم route_counts_cleaned می خواند. و row شامل اطلاعات یک مسیر حمل و نقل هست. Scattergeo نوع نمودار جغرافیایی که می تونه نقاط یا خطوط رو روی نقشه رسم کند. lon ، لیست طول جغرافیایی مبدأ و مقصد را برای هر سطر ذخیره می کند و lat ، لیست عرض جغرافیایی مبدأ و مقصد برای هر سطر ذخیره می کند. mode='lines' ، یک خط بین این دو مختصات رسم می کند. hoverinfo='text' ، وقتی موس رو می بری روی خط، متن دلخواه نمایش داده می شود. text() ، متن شامل نام شهر مبدأ و نام شهر مقصد (تعداد دفعات حمل) هست.

نوع projection_type='mercator' scope='asia' یعنی در محدوده ای آسیا نقشه تنظیم می شود. نمایش نقشه را مشخص می کند. showland=True خشکی ها را نمایش می دهد. lonaxis و lataxis

نقشه در محدوده عرض و طول جغرافیایی ایران تنظیم می شود. center مرکز نقشه روی ایران می گیرد. حذف حاشیه های اضافه برای استفاده کامل از فضا با استفاده از margin.

نقشه مسیر های حمل و نقل

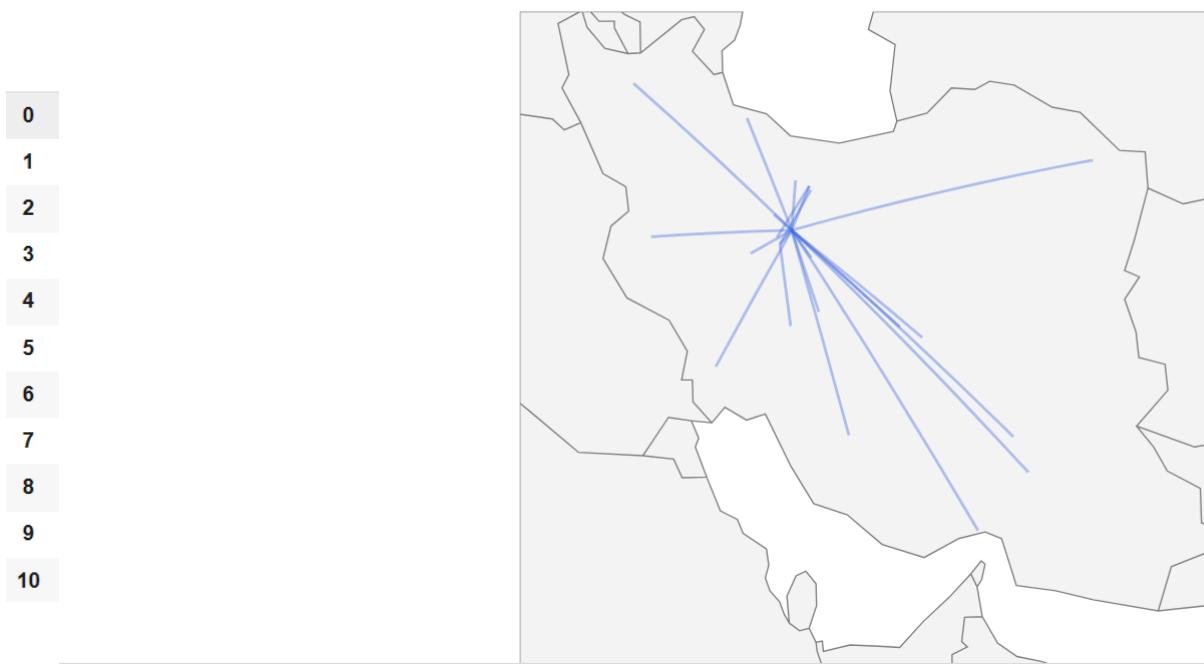


کل مسیرهایی که در این دیتافریم هست ، نمایش داده شده است. همانطور که مشاهده می شود ، مرکزیت اکثر مسیرها به سمت قم و تهران است. همانطور که دیده می شود حمل و نقل در اکثر مناطق کشور وجود دارد و کالا و محصولات را به دست مشتران رسانده شده است.

برای اینکه مسیرهای پرتردد را بر روی نقشه مشاهد کنیم ، میتوانیم روی داده های دیتافریم route_counts_cleaned فیلتر بگذاریم . یعنی بگوییم هر مسیری را نشان بده که تعداد تردد از آن مسیر بیشتر از ۵۰۰۰ باشد. و مثل نقشه بالا رسم می کنیم :

تعداد ۲۲ مسیر وجود دارند که تعداد تردد آن ها از ۵۰۰۰ بالاتر و بیشتر است . و همانطور که مشاهده می شود مرکزیت شهرهای مبدأ اطراف قم (شهرستان های آن) و خود قم قرار دارد.

نقشه مسیرهای پرتردد حملونقل



بررسی پرکاربردترین مبادی و مقاصد

برای اینکه متوجه شویم کدام مبادی پرکاربردترین (پرترکار ترین) مبادی هستند، باید تعداد تکرار همه‌ی مبادی در دیتاست را بدست آوریم. می‌توانیم از دو روش این کار را انجام دهیم گزینه اول تجمعی براساس ستون مبدا است و گزینه دوم استفاده از تابع `value_counts()`، ما در اینجا از روش دوم استفاده نموده ایم؛ سپس حاصل این عملیات دو ستون است که در دیتافریم جدیدی به نام `origin_counts` قرار می‌دهیم و نام جدیدی برای ستون‌های میگذاریم؛ نام ستون شهرهای مبدا را `City` و نام ستون شمارش تعداد مبدا را `Origin Count` می‌گذاریم. در خروجی همچین دیتافریم را داریم :

برای شهر مقصد هم، همین کارها را انجام می‌دهیم و نام دیتافریم ایجاد شده را `destination_counts` و نام ستون هایش را به ترتیب `City` و `Destination Count` می‌گذاریم. حال می‌توانیم این دو دیتافریم را با هم ادغام (`merge`) کنیم و دو دیتافریم را انتخاب می‌کنیم و ستونی (`City`) که می‌خواهیم براساس آن ادغام را انجام دهیم هم مشخص می‌کنیم. سپس نوع

ادغام را هم مشخص می کنیم و پارامتر 'how='outer' قرار می دهیم این یعنی اگر شهری فقط در یکی از دو جدول باشد (فقط مبدا یا فقط مقصد باشد)، باز هم حذف نمی شود. تابع fillna(0) هم برای ستون هایی که مقدار ندارند صفر می گذارد. نام دیتافریم جدید را city_counts می گذاریم. ستونی ایجاد می کنیم تا دو مقادیر دو ستون Destination Count و Origin Count را جمع کند و برای هر سطر بدست آورد و تعداد کل شمارش ها را محاسبه کند؛ نام ستون جدید را Total Count می نامیم. دیتافریم city_counts را براساس ستون Total Count به صورت نزولی مرتب می کنیم.

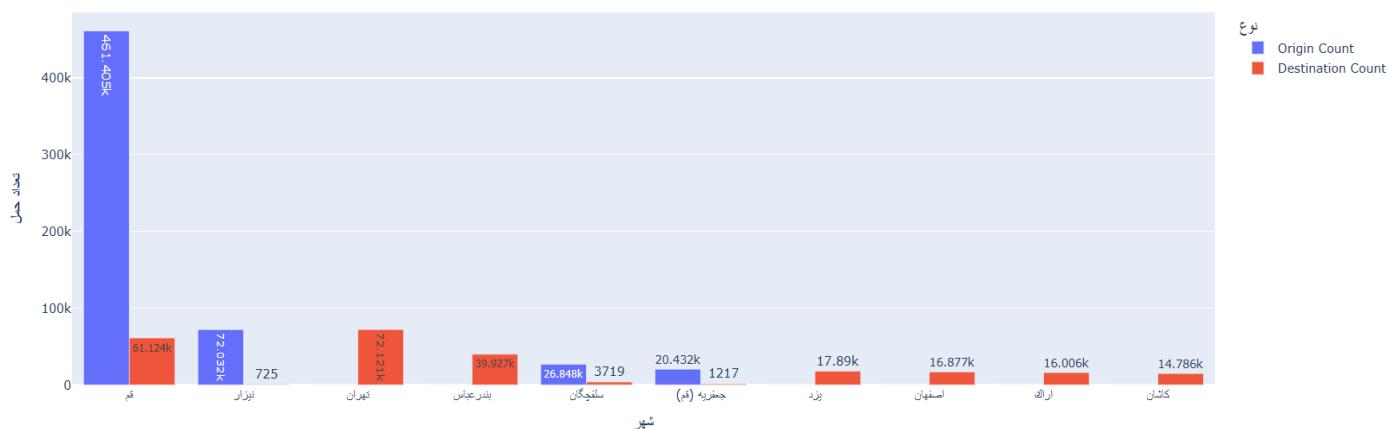
۱۰ تا از سطرهای این دیتافریم را که بیشترین مقدار Total Count را دارند در دیتافریم جدیدی به نام top_cities ذخیره می کنیم و نمودار این ۱۰ سطر را رسم می نماییم.

```
fig = px.bar(top_cities,
              x='City',
              y=['Origin Count', 'Destination Count'],
              title='پر تکرارترین شهرها به عنوان مبدأ و مقصد',
              barmode='group',
              labels={'value': 'تعداد حمل', 'variable': 'نوع'},
              text_auto=True)
fig.update_layout(xaxis_title='شهر', yaxis_title='تعداد')
fig.show()
```

طبق کدهای رو به رو نمودار زیر را رسم می کنیم . در قسمت محور ۷ هر دو ستون Destination Count و Origin Count را در مجموع می دهیم. چون میخواهیم دو ستونی که در

محور ۷ قرار دادیم به صورت مجزا از هم نمایش داده شوند پارامتر 'barmode='group'' قرار می دهیم. نمودار به صورت زیر خواهد بود :

پر تکرارترین شهرها به عنوان مبدأ و مقصد



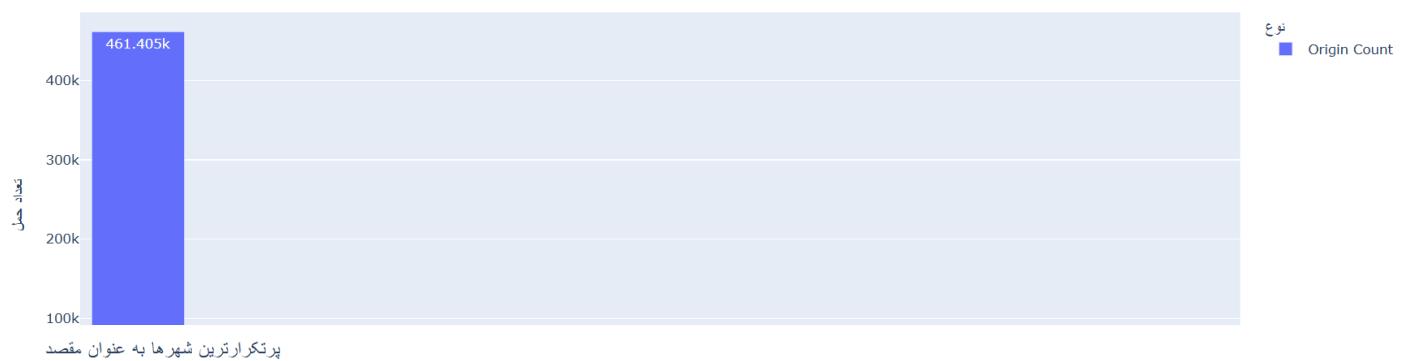
عنوان این نمودار را پر تکرارترین شهرها به عنوان مبدأ و مقصد گذاشتیم اما درستش آن بود که قرار

دهیم مجموع پر تکرار ترین مبادی و مقاصد با هم برای این نمودار هستند. بالاترین مقدار برای این تعداد تکرار مبدا ، شهر قم است با تعداد تکرار ۴۶۱.۴۰۵ هزار تکرار. پس شهر قم در مجموع پر تکرار ترینی شهر مبدا و مقصد بوده یعنی ما تعداد تکرار شهر قم به عنوان مبدا و تعداد تکرار شهر قم به عنوان مقصد با هم جمع کردیم اولین شهر است.

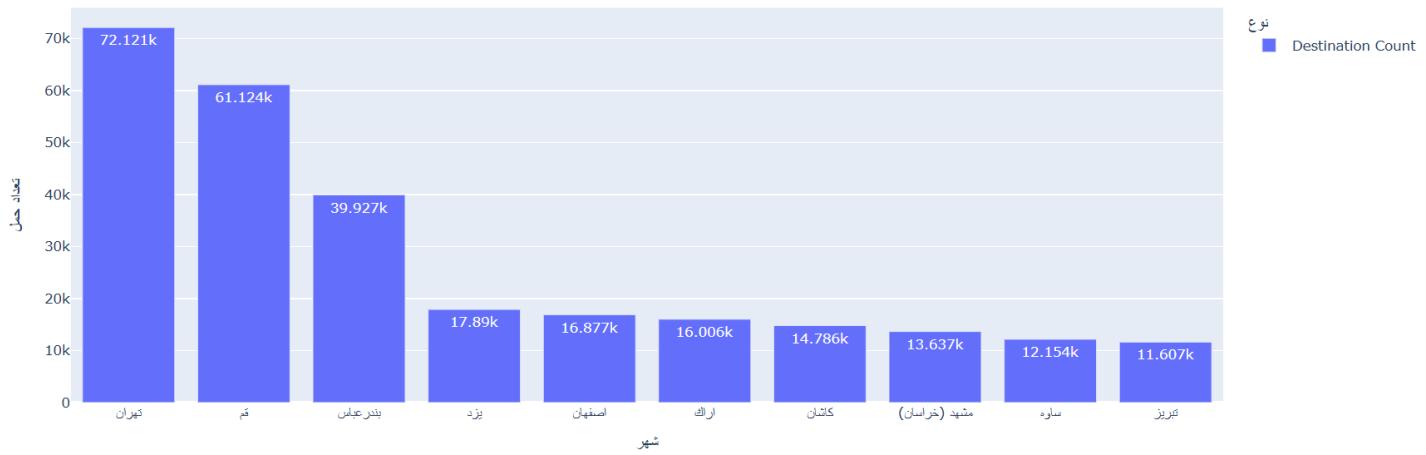
حالا برای اینکه متوجه شویم چه شهرهایی به عنوان مبدا و چه شهرهایی به عنوان مقصد به صورت مجزا پر تکرار بوده اند نمودارشان را رسم می کنیم اما قبل از آن ، باید دیتا فریم origin_counts را براساس ستون Origin Count به صورت نزولی مرتب کنیم و سپس ۰۱ شهر پر تکرار را در دیتا فریم جدید به نام top_origin ذخیره کنیم. سپس نمودار این دیتا فریم را رسم می کنیم ، در محور X نام شهر و در محور Y ستون تعداد تکرار را می گذاریم.

برای مقصد هم همین روند را انجام می دهیم و نمودارهای هر دو دیتا فریم به صورت زیر خواهد بود:

پر تکرار ترین شهرها به عنوان مقصد



پر تکرار ترین شهرها به عنوان مقصد



همانطور که در نمودار پر تکرار ترین شهرها به عنوان مبدا می بینید ، شهر قم به عنوان پر تکرار ترین مبدا و پس از آن نیزار . سلفچگان قرار دارند. و در نمودار مرتبط با پر تکرار ترین شهرها به عنوان مقصد ، شهر تهران به عنوان پر تکرار ترین مقصد و پس از آن شهر قم و بندرعباس قرار دارند.

تحلیل هزینه به ازای کیلومتر برای هر مسیر

زیرا گفته شده است «هزینه به ازای کیلومتر » ، پس باید با استفاده از دو ستون Calculateion fare و Distance از دیتاست transport_df ، ستون جدیدی به عنوان هزینه به ازای هر کیلومتر ایجاد کنیم. با تقسیم ستون کرایه بر مسافت این ستون ساخته می شود و نام ستون را Fare_per_km و در دیتاست transport_df ذخیره می کنیم. و چون در صورت مسئله گفته شده «برای هر مسیر» پس باید ستون مسیر را همانطور که قبل از قسمت های قبلی ایجاد کرده بودیم ، بسازیم.

```
#(origin -> destination)
transport_df['route_col'] = transport_df['Name of the city of origin'] + ' → ' + transport_df['The name of the destination city']
```

برای اینکه متوجه بشوی برای هر مسیر جمعا هزینه به ازای هر کیلومتر چقدر است ، تجمعی انجام می دهیم و میانگین ستون Fare_per_km برای هر مسیر را بدست می آوریم ، همچین می توانیم تعداد حمل را برای هر مسیر براساس ستون Fare_per_km بدست بیاوریم. میانگین ستون های دیگر مانند مسافت و کرایه را هم به عنوان اطلاعات تكمیلی بدست می آوریم و مثل همیشه route_cost() را بکار می بریم نام دیتافریم را reset_index() می گذاریم. نام ستون های بدست

```
route_cost.columns=[  
    'route_col',  
    'Mean Distance',  
    'Mean Calculateion fare',  
    'Avg Fare per km',  
    'Count'  
]
```

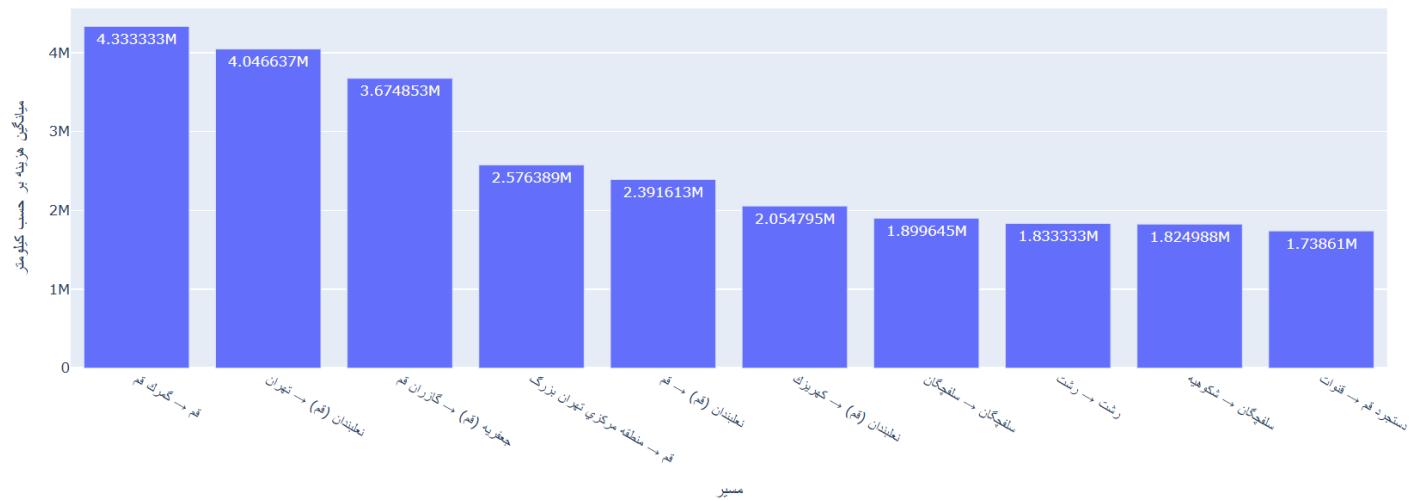
آمده را هم تغییر می دهیم. به ترتیب اسمای ستون ها هستند :

سپس دیتا فریم route_cost را براساس ستون Avg Fare per km را مرتب می کنیم و ۱۰ سطر اول آن را در

دیتافریم جدید top_expensive_routes دستخراج و نمودارشان را رسم کنیم. نمودار زیر نشان دهنده می مسیرهایی هزینه بر حسب کیلومتر را استخراج و نمودارشان را رسم کنیم. نمودار زیر نشان دهنده می مسیرهایی با بیشترین هزینه بر حسب کیلومتر است :

همانطور که دیده می شود مسیر قم-گمرک قم بیشترین مقدار میانگین هزینه بر حسب کیلومتر را با ۴.۳۳۳۳ میلیون دارد که تعداد حمل حالا هم فقط ۱ بوده است. این مسیر یک مسیر خیلی کوتاه است و کرایه به ازای هر کیلومتر خیلی بالاست.

مسیرهایی با بیشترین کرایه محاسباتی بر حسب کیلومتر



دومین گران ترین هزینه به ازای هر کیلومتر ، مسیر نعلبندان(قم)-تهران است با هزینه ۴۰۴۶۶۳ میلیون برای هر کیلومتر و این مسیر هم مانند مسیر قبلی مسیر کوتاهی است.

خیلی از این مسیرها کوتاه هستند و ممکن است بخارطه وزن کالا و سایر ویژگی های دیگر این هزینه اعمال شده باشد.

ارزان ترین مسیرها را هم با گذاشتن محدودیت برای ستون $5000 > \text{Avg Fare per km}$ بدست

می آوریم.

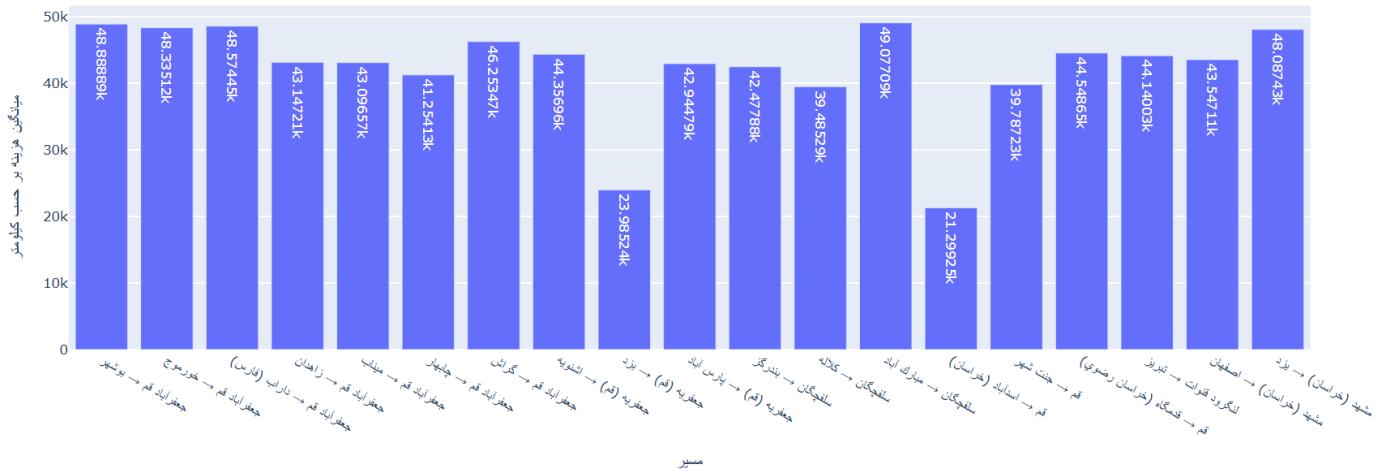
```

▶ # مسیرهایی با هزینه کمتر از آستانه مشخص
cheap_routes = route_cost[route_cost['Avg Fare per km'] < 50000]
cheap_routes

```

حدود ۱۹ شهر هزینه به ازای کیلومترشان کمتر از ۵۰۰۰ است . نمودار این مسیرها را هم رسم می کنیم.

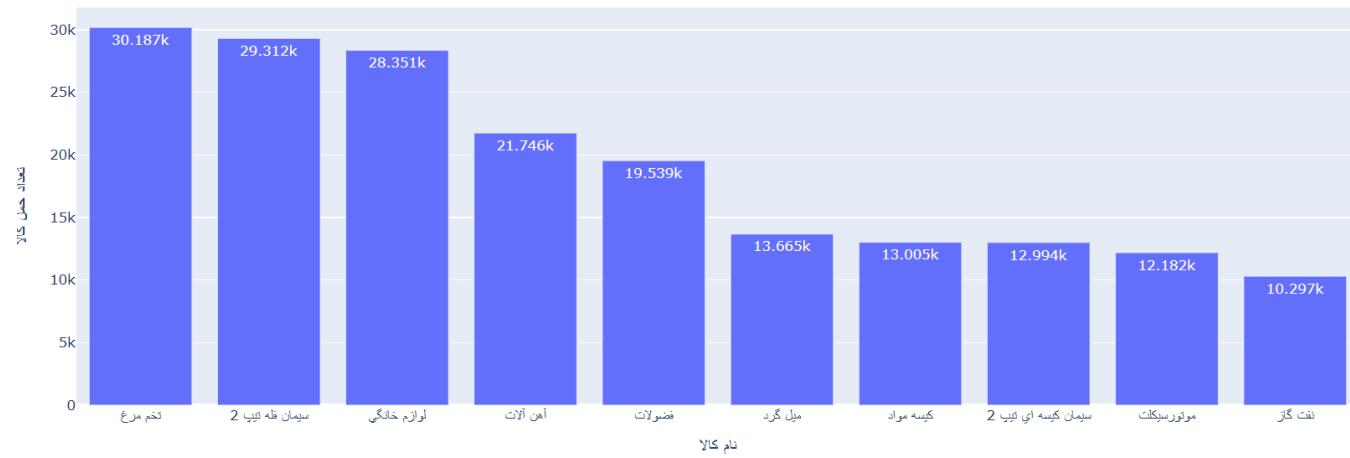
با توجه به نمودار ، کمترین مقدار برای ستون میانگین هزینه برحسب کیلومتر ، مسیر قم – اسد آباد (خراسان) است با $21.29925k$ است. همانطور که مشاهده می کنید این نمودار یعنی کمترین هزینه ها نسبت به نمودار بیشترین هزینه برای مسیرها ، شامل مسیرهایی طولانی تر هستند برای مثال جعفرآباد قم – زاهدان و جعفرآباد قم – چابهار مسیری طولانی است .



بیشترین حمل مربوط به چه کالایی است ؟

با توجه به اینکه سوال این است که « بیشترین حمل مربوط به چه کالایی است ؟ » باید تعداد حمل هر محصول را بدست بیاوریم. براساس ستون نام کالا (Product name) در دیتابیست `transport_df` ، تجمعیت انجام می دهیم و تعداد سطرهای مربوط به هر کالا را می شماریم و با اینکار تعداد حمل هر کالا را بدست می آوریم. نام ستون بدست آمده را با تابع `reset_index()` در پارامتر `name` می گذاریم. نام دیتابیفریم را `product_count` می گذاریم. سپس دیتابیفریم را براساس ستون `count` نزولی مرتب می کنیم و ۱۰ سطر اول را در دیتابیفریم دیگری به نام `top_carried_products` ذخیره می کنیم. حالا ما ۱۰ کالا با بیشترین حمل را بدست آورده ایم و کافیست بر روی نمودار این دیتابیفریم را رسم کنیم. محور `x` نام کالا و محور `y` را به ستون `count` اختصاص می دهیم.

نمودار کالاهایی که بیشترین حمل را داشته اند



بیشترین حمل کالا مربوط به تخم مرغ با تعداد حمل ۳۰.۱۷۸k و رتبه ۵ دوم مربوط به سیمان فله تیپ ۲ با تعداد حمل ۲۹.۳۱۲k و پس از لوازم خانگی و آهن آلات و ... قرار دارند.

مقایسه هزینه حمل براساس نوع کالا

برای انجام این تحلیل می خواهیم بررسی کنیم که کدام کالاهای

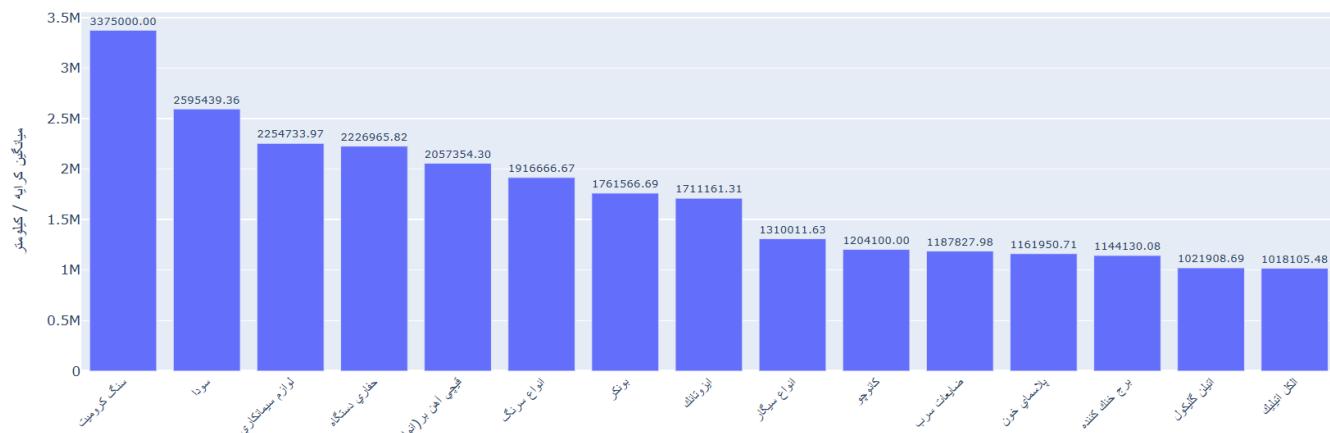
۱. گرانتر حمل می شوند (میانگین کرایه بالاتری دارند)
۲. نسبت به مسافت، هزینه بیشتری دارند (کرایه به ازای کیلومتر بالاتری دارند)
۳. تعداد دفعات حملشان چقدر است؟ (حجم حمل)

کرایه محاسباتی به ازای کیلومتر را محاسبه می کنیم (مشابه بخش های قبل). گروپ با بر روی ستون محصول می زنیم و میانگین کرایه ، میانگین کرایه محاسباتی (هزینه) به ازای هر کیلومتر ، rename() میانگیم مسافت ، و تعداد حمل برای هر کالا را با ستون نام کالا بدست می آوریم و با تابع () name ستون ها را تغییر می دهیم و با تابع reset_index() شماره سطرها را به عنوان ایندکس قرار می دهیم. و نام دیتا فریم را product_fare_stats می گذاریم.

این دیتافریم را براساس ستون Mean_Fare_per_km به صورت نزولی مرتب می کنیم و ۱۵ تا سطر اول یعنی کالاهایی که هزینه حملشان به ازای هر کیلومتر از همه بیشتر است را در دیتافریم دیگری به نام top_products ذخیره می کنیم . نمودار این دیتافریم را رسم می کنیم.

محور x ، ستون نام محصول و محور y ستون Mean_Fare_per_km است و بقیه اطلاعات برای ستون های دیگر را در بخش hover_data قرار می دهیم تا با قرار دادن موس بر روی هر کدام از محصول بر روی نمودار اطلاعاتش ظاهر شود. نمودار به صورت زیر است :

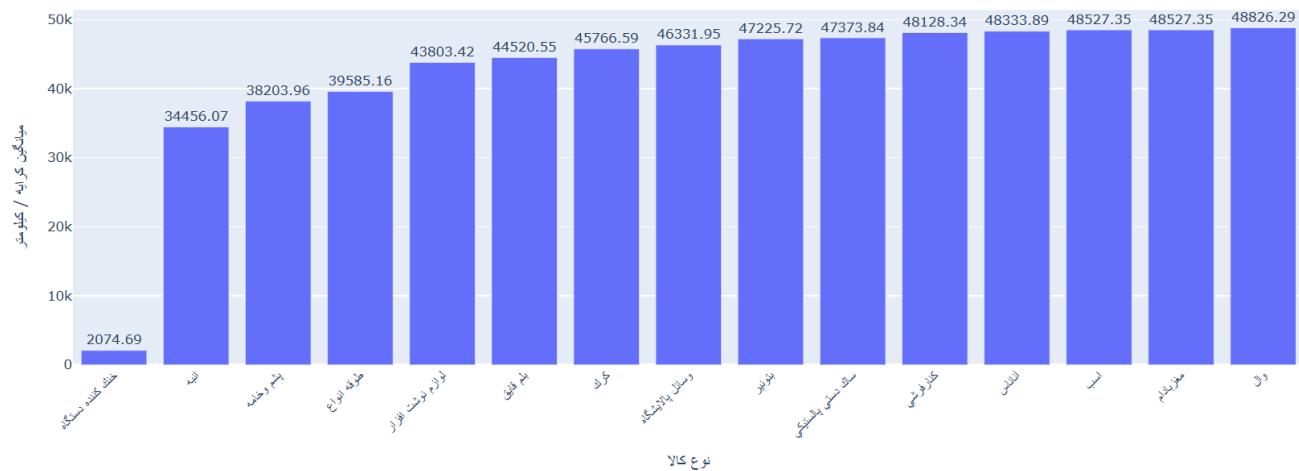
مقایسه ۱۵ کالا که بیشترین میانگین کرایه به ازای کیلومتر را داشته اند



همانطور که مشاهده می کنید نمودار برای ۱۵ کالایی که گرانترین و بالاترین هزینه حمل به ازای هر کیلومتر را داشته اند به تصویر کشیده شده است. اولین کالا با هزینه ۳۳۷۵۰۰۰ میلیون در صدر نمودار قرار دارد که میانگین مسافت آن هم حدود ۸۰ کیلومتر است و بعد از آن سودا و لوازم سیمانکاری با هزینه های ۲۵۹۵۴۳۹ و ۲۲۵۴۷۳۳ قرار دارند. مشابه نمودار بالا ، نموداری برای ۱۵ کالایی که کمترین هزینه و کرایه به ازای هر کیلومتر را داشته اند را هم رسم کرده ایم که به صورت زیر است :

کالایی که کمترین هزینه را داشته است دستگاه خنک کننده با ۱ بار حمل است و میانگین مسافتی ۴۸۲ کیلومتر است . بعد از آنکه کمترین هزینه را دارد.

مقایسه 15 کالا که کمترین میانگین کرایه به ازای کیلومتر را داشته اند



تشخیص کالاهای سنگین ولی کم بازده از نظر اقتصادی

هدف از این تحلیل این است که :

۱. یافتن کالاهایی که میانگین وزن حمل بالایی دارند
۲. اما میانگین کرایه دریافتی بر کیلومتر پایین دارند.

پس دوباره باید هزینه (کرایه محاسباتی) به ازای هر کیلومتر را محاسبه کنیم.

```
تجمعی آماری داده ها بر اساس نوع کالا
heavy_low_profit_products = transport_df.groupby('Product name').agg({
    'Weight' : 'mean',
    'Calculateion fare': 'mean',
    'Distance': 'mean',
    'Fare_per_km': 'mean',
    'Product name': 'count'
}).rename(columns={
    'Weight' : 'mean Weight',
    'Calculateion fare': 'mean Calculateion fare',
    'Distance': 'mean Distance',
    'Fare_per_km': 'mean_Fare_per_km',
    'Product name': 'count'
}).reset_index()
```

مانند قسمت های قبل تجمعی انجام می دهیم و از ستون های مورد نیاز میانگین می گیریم. نام دیتا فریم جدید را heavy_low_profit_products می گذاریم.

تعریف معیار برای «سنگین» و «کم بازده» بودن تعریف می کنیم . می توانیم از میانگین گیری برای درنظر گیری معیار استفاده کنیم. استفاده از «میانگین میانگینها» منطقی است اما حساس به

است. اگر توزیع دارای مقادیر بسیار بزرگ یا کوچک باشد، شاید میانه یا قطکها outlier مناسب‌تر باشند. اما ما همان میانگیم را برای معیار انتخاب می‌کنیم.

حال فیلتر کالاهای «سنگین ولی کمبازده» را انجام می‌دهیم.

▶ **فیلتر بر اساس شرایط: وزن بالا، کرایه بر کیلومتر پایین**
محصولات ناکارآمد (همان کالاهای سنگین ولی کم بازده از نظر اقتصادی)

```
inefficient_products = heavy_low_profit_products[ # (heavy_low_profit_products['mean Weight'] > avg_weight) & (heavy_low_profit_products['mean_Fare_per_km'] < avg_fare_per_km)]
```

کد بالا یک فیلتر بولی روی جدول ایجاد می‌کند؛ فقط کالاهایی را نگه می‌دارد که هم وزن بالاتر از میانگین دارند و هم کرایه بر کیلومترشان پایین‌تر از میانگین است. نتیجه inefficient_product جدولی از همان ستون‌ها ولی فقط شامل کالاهای «سنگین و کمبازده» است.

سپس جدول و دیتافریم بالا را براساس ستون mean Weight نزولی مرتب سازی می‌کنیم.

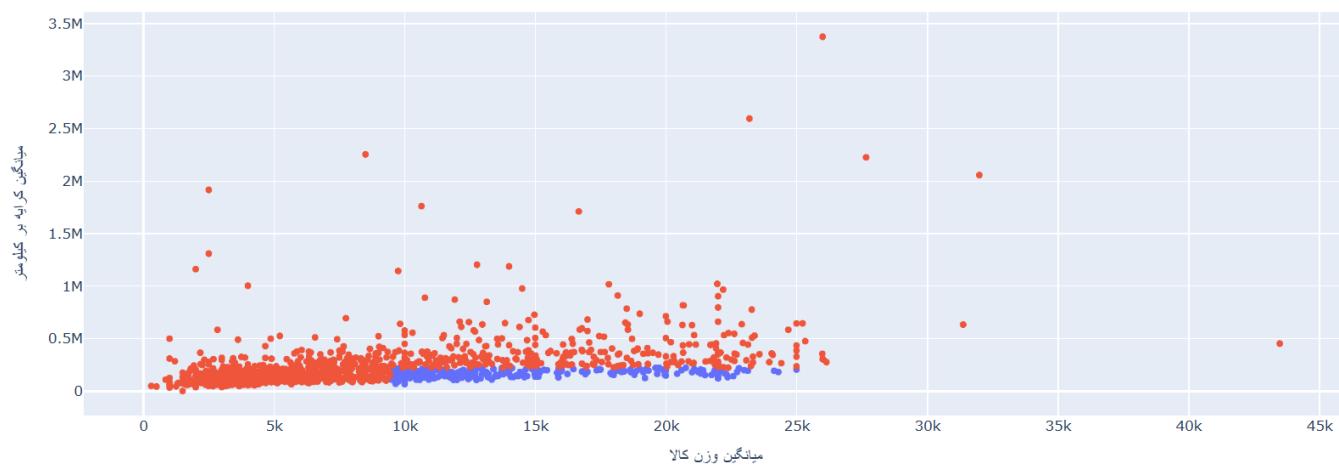
رسم نمودار :

```
fig = px.scatter(heavy_low_profit_products,
                  x='mean Weight',
                  y='mean_Fare_per_km',
                  # text='Product name',
                  title='تحلیل کالاهای سنگین و کمبازده از نظر اقتصادی',
                  labels={
                      'color' : 'کالا',
                      'Product name' : 'نام کالا',
                      'mean Weight' : 'میانگین وزن کالا',
                      'mean Calculateion fare' : 'میانگین کرایه محاسباتی',
                      'mean Distance' : 'میانگین مسافت',
                      'mean_Fare_per_km': 'میانگین کرایه بر کیلومتر',
                      'count' : 'تعداد محمل'
                  },
                  hover_data ={
                      'Product name' : True ,
                      'mean Weight': ':.2f' ,
                      'mean Calculateion fare' : ':.2f' ,
                      'mean Distance' : ':.2f' ,
                      'mean_Fare_per_km': ':.2f' ,
                      'count' : True
                  },
                  color=(heavy_low_profit_products['mean_Fare_per_km'] < avg_fare_per_km) & (heavy_low_profit_products['mean Weight'] > avg_weight))
fig.update_traces(textposition='top center')
fig.show()
```

نمودار ، داده‌ها را از heavy_low_profit_products می‌گیرد (یعنی همه کالاهای را رسم می‌کند، نه فقط inefficient_products). محور x میانگین وزن کالا و محور y میانگین کرایه/کیلومتر را نشان می‌دهد. ستون‌هایی که وقتی موس را روی نقطه می‌بری نمایش داده می‌شوند و فرمت آن‌ها ':.2f' برای نمایش دو رقم اعشار است. color یک Expression بولی

(دادهای ساخته شده و به Plotly پاس داده شد color) این بولی را تبدیل به دو رنگ می کند: رنگ آبی نشان دهنده مقدار بولی True و مقدار قرمز نشان دهنده مقدار بولی False است. یعنی رنگ های آبی در این بازه صدق می کنند و کالاهای سنگین ولی کم بازده از نظر اقتصادی هستند ولی داده هایی با رنگ قرمز کالاهایی هستند که کم بازده از نظر اقتصادی نیستند و سنگین هم نیستند و یا در یکی از این شرط ها صدق نکرده اند. نمودار به صورت زیر است :

تحلیل کالاهای سنگین و کمبازده از نظر اقتصادی



تحلیل نمودار :

حال نمودار دیتا فریم inefficient_product هم رسم می نماییم. این دیتا فریم ۲۸۷ تا سطر دارد یعنی ۲۷۴ کالا سنگین ولی بازده اقتصادی ندارند. این ۲۸۷ ستون را رسم می کنیم. کالاهایی که سنگین و کمبازده هستند (وزن بالاتر از میانگین، کرایه بر کیلومتر پایین تر از میانگین) با رنگ آبی نمایش داده شدند. کالاهایی که در سمت راست پایین نمودار قرار گرفتند، هم سنگین هستند و هم کرایه بر کیلومتر پایینی دارند و ناکارآمدترین بخش حمل هستند. کالاهای سمت راست بالا سنگین هستن ولی بازده اقتصادی خوبی دارند. کالاهای سمت چپ پایین سبک و کمبازده هستند و از نظر سوددهی هم مهم ولی به خاطر سبک بودن فشار زیادی به هزینه حمل نمی آورند.

بر روی محور X ستون نام کالا و بر روی محور y ، mean_Fare_per_km قرار می دهیم. اطلاعات تمام ستون های دیگر را در قسمت هاور نمایش می دهیم. در آخر نمودار مانند زیر است :

ستون‌ها به ترتیب میانگین کرایه بر کیلومتر را برای کالاهای ناکارآمد نشان می‌دهند. هرچی ارتفاع ستون کمتر باشد، اون کالا کم‌بازده‌تر است. این نمودار دیگر تصویری کلی از همه کالاهای نمی‌دهد، بلکه بر روی مشکل‌دارترین بخش‌ها تمرکز دارد. این نمودار برای اولویت‌بندی اقدامات اصلاحی بسیار کاربردی است، چون به وضوح مشخص کرده است کدام کالا بیشترین نیاز به بازنگری قیمت یا کاهش هزینه حمل را دارد.

تحلیل کالاهای سنگین و کم‌بازده از نظر اقتصادی

