نام اعضای گروه

ریحانه سادات میرعربشاهی

متین احمدی

1. تحلیل اکتشافی داده (Exploratory Data Analysis)

در این پروژه هدف ما پیش‌بینی دلیل سفر کاربران بر اساس اطلاعات مربوط به خرید و بلیط است. کاربران بیلیتیو ممکن است بلیط را با هدف کاری (Work) یا تفریحی (Int) رزرو کنند. دانستن دلیل سفر می‌تواند به شخصی‌سازی پیشنهادها، پیام‌ها و کمپین‌های بازاریابی کمک کند؛ برای مثال پیشنهاد بیمه/هتل/تور، یا ارائه پیشنهادهای مرتبط با سفرهای کاری مانند خدمات سازمانی. بنابراین مسئله را می‌توان یک مسئله‌ی یادگیری نظارت‌شده (Supervised Learning) از نوع طبقه‌بندی (Classification) در نظر گرفت که در آن متغیر هدف TripReason و ورودی‌ها مجموعه‌ای از ویژگی‌های زمانی، مالی و رفتاری کاربر و سفارش هستند.

داده‌ی آموزشی شامل حدود ۱۰۰ هزار رکورد است و برای هر بلیط اطلاعاتی مانند زمان ثبت، زمان حرکت، وضعیت کنسلی، قیمت، تخفیف، مبدا/مقصد، نوع وسیله نقلیه و... ارائه شده است. داده‌ی تست نیز ساختاری مشابه دارد اما ستون هدف TripReason را ندارد و باید مقدار آن پیش‌بینی شود.

چالش‌های اصلی این مسئله عبارت‌اند از:

ویژگی‌های زمانی (Created, DepartureTime, CancelTime) زمان‌ها به شکل خام معمولاً برای مدل قابل استفاده نیستند، بنابراین باید به ویژگی‌های معنی‌دار تبدیل شوند (مثل فاصله زمانی تا حرکت یا ماه حرکت).

ویژگی‌های دسته‌ای با تعداد مقادیر زیاد (مثل From/To/Vehicle/ReserveStatus و...) این ستون‌ها باید به شکل عددی تبدیل شوند (Encoding) تا برای مدل قابل استفاده باشند.

وجود مقادیر گمشده و داده‌های غیرمنطقی مثل VehicleClass خالی، تخفیف‌های منفی، یا قیمت‌های نامعتبر که باید پاکسازی شوند تا باعث افت کیفیت مدل نشوند.

ساختار سفارش و چند بلیطی بودن یک خرید هر BillID ممکن است چند بلیط داشته باشد؛ بنابراین استخراج الگوهای سفارش می‌تواند اطلاعات مهمی درباره هدف سفر بدهد.

مجموعه داده آموزشی شامل ۱۰۱۰۱۷ سطر است که در جدول زیر، توضیحات هر ستون آمده است.

مجموعه داده آزمایش نیز مانند مجموعه آموزش است با این تفاوت که ستون `TripReason` که متغیر هدف مسئله است را در خود ندارد. مجموعه داده آزمایش ۴۳۲۹۳ سطر دارد.

برای ارزیابی عملکرد مدل از معیار F1-Score استفاده می‌شود. F1 ترکیبی از Precision و Recall است و زمانی مناسب است که:
عدم توازن کلاس‌ها وجود داشته باشد (مثلاً یکی از دلایل سفر بیشتر رخ دهد)، یا هزینه‌ی خطا در هر دو جهت مهم باشد (هم شناسایی درست سفر کاری و هم تفریحی اهمیت دارد). به همین دلیل F1 معیار مناسبی برای سنجش کیفیت مدل در این مسئله است و علاوه بر دقت، به «تعادل در پیش‌بینی‌ها» نیز توجه می‌کند.

برای رسیدن به یک مدل قابل دفاع، فرآیند کار به صورت زیر طراحی شده است:

1) تحلیل اکتشافی داده (EDA)

ابتدا داده بررسی می‌شود تا:
توزیع کلاس‌های TripReason مشخص شود، میزان مقادیر گمشده و ستون‌های مشکل‌دار شناسایی شود، رفتار ویژگی‌های عددی مانند Price و CouponDiscount تحلیل شود، و روابط احتمالی بین ویژگی‌ها و دلیل سفر پیدا شود. این مرحله به ما کمک می‌کند پیش‌پردازش‌ها و مهندسی ویژگی‌ها را آگاهانه انتخاب کنیم.

2) پیش‌پردازش و پاکسازی

اقدامات اصلی شامل:
حذف ستون‌های شناسنامه‌ای یا کم‌اثر (مثل هش‌ها و شناسه‌ها) برای جلوگیری از نویز و حفظ حریم خصوصی، مدیریت مقادیر گمشده (مثلاً پر کردن VehicleClass با مقدار مد)، اصلاح و پاکسازی ویژگی‌های مالی (حذف/اصلاح قیمت‌های نامعتبر و تخفیف‌های غیرمنطقی)، و نرمال‌سازی ویژگی‌های عددی مثل Price برای کمک به مدل‌های حساس به مقیاس.

3) مهندسی ویژگی (Feature Engineering)

برای افزایش قدرت مدل، ویژگی‌های زیر ساخته می‌شود:
TicketPerOrder: تعداد بلیط‌های ثبت‌شده در هر خرید (BillID) چون سفرهای خانوادگی یا

گروهی معمولاً چند بلیط دارند و ممکن است با نوع سفر مرتبط باشد.
family: تشخیص سفارش‌هایی که هم مسافر زن و هم مرد دارند این ویژگی با هدف捕 کردن الگوی «سفر خانوادگی» ساخته شده است که می‌تواند به نوع سفر نزدیک باشد.
Departure_Created: فاصله زمانی بین ثبت بلیط و زمان حرکت چون سفر کاری ممکن است نزدیک‌تر به زمان حرکت رزرو شود یا برعکس، این فاصله می‌تواند سیگنال مهمی باشد.
DepartureMonth: ماه حرکت فصل‌ها و تعطیلات می‌توانند بر احتمال تفریحی بودن سفر اثرگذار باشند.
Discount: وجود یا عدم وجود تخفیف استفاده از تخفیف ممکن است الگوی رفتاری خاصی در سفرهای تفریحی یا کاری ایجاد کند.

4) مدل‌سازی و مقایسه چند مدل
در ادامه چند مدل رایج طبقه‌بندی آموزش داده می‌شود تا بهترین گزینه انتخاب شود، از جمله:
Logistic Regression
KNN
Random Forest
XGBoost
این مقایسه به ما کمک می‌کند مدل نهایی را بر اساس عملکرد واقعی انتخاب کنیم، نه صرفاً بر اساس حدس.

در این بخش توزیع کلاس هدف TripReason بررسی می‌شود. این نمودار خیلی مهم است چون اگر کلاس‌ها نامتوازن باشند (مثلاً تعداد سفرهای تفریحی خیلی بیشتر از کاری باشد)، انتخاب معیار ارزیابی (مثل F1)، استفاده از Stratify در split و حتی تنظیم class_weight یا روش‌های بالانس‌سازی اهمیت پیدا می‌کند. همچنین از همین‌جا یک تصویر اولیه از سختی مسئله می‌گیریم.

در این مرحله، توزیع کلاس‌های متغیر هدف **TripReason** مورد بررسی قرار گرفت. نتایج نشان می‌دهد که تعداد نمونه‌های کلاس **Work** (سفر کاری) بیشتر از کلاس **Int** (سفر تفریحی) است و نسبت تقریبی آن‌ها به ترتیب حدود ۵۵٪ و ۴۵٪ می‌باشد.

این اختلاف بیانگر وجود **عدم توازن خفیف در داده‌ها** است. هرچند شدت این عدم توازن بالا

نیست، اما در فرآیند مدل‌سازی باید مورد توجه قرار گیرد؛ زیرا در صورت بی‌توجهی، ممکن است مدل به سمت کلاس اکثریت متمایل شود و عملکرد آن به‌درستی ارزیابی نشود.

به منظور کاهش اثر این موضوع، در مرحله تقسیم داده‌ها از روش **Stratified Train-Test Split** استفاده می‌شود تا نسبت کلاس‌ها در داده‌های آموزش و اعتبارسنجی حفظ گردد. همچنین در ارزیابی عملکرد مدل، علاوه بر Accuracy، از معیارهایی مانند **F1-score** استفاده خواهد شد تا تعادل بین Precision و Recall در نظر گرفته شود.

در مجموع، با وجود عدم توازن خفیف، با به‌کارگیری روش‌های مناسب در تقسیم داده و ارزیابی، انتظار می‌رود تأثیر این موضوع بر نتایج نهایی کنترل شود.

در این بخش بررسی می‌کنیم آیا قیمت پرداختی می‌تواند سیگنال مناسبی برای تشخیص نوع سفر باشد یا نه. چون Price به‌تنهایی ممکن است با تخفیف‌ها گمراه‌کننده شود، ابتدا قیمت نهایی را به‌شکل FinalPrice = Price - CouponDiscount می‌سازیم. از آنجایی که قیمت‌ها معمولاً توزیع کج (Skewed) دارند، از log(1+FinalPrice) استفاده می‌کنیم تا مقایسه‌ی دو کلاس در نمودار جعبه‌ای معنی‌دارتر شود.

در این بخش، توزیع متغیر **FinalPrice** بین دو کلاس **Work** و **Int** با استفاده از نمودار Boxplot بررسی شد. به منظور کاهش اثر کشیدگی توزیع و بهبود خوانایی، از تبدیل **log(1 + FinalPrice)** استفاده گردید.

نتایج نشان می‌دهد که میانه قیمت‌ها در دو کلاس بسیار نزدیک به یکدیگر است و پراکندگی داده‌ها (بر اساس IQR و دامنه خطوط whisker) نیز شباهت قابل توجهی دارد. این موضوع بیانگر آن است که متغیر قیمت نهایی، حداقل به‌تنهایی و در این مقیاس، قدرت تفکیک بالایی بین دو کلاس ندارد.

همچنین وجود تعداد قابل توجهی داده پرت در هر دو کلاس نشان‌دهنده توزیع دم‌کلفت قیمت‌هاست. علاوه بر این، مشاهده یک مقدار بسیار نزدیک به صفر در یکی از نمونه‌های

کلاس **Int** حاکی از وجود رکوردی با قیمت نهایی تقریباً صفر است که بررسی بیشتر آن از نظر صحت داده توصیه می‌شود.

در مجموع، اگرچه قیمت نهایی اطلاعات مفیدی ارائه می‌دهد، اما به تنهایی شاخص تعیین‌کننده‌ای برای تمایز بین اهداف سفر محسوب نمی‌شود.

اینجا رفتار رزرو از نظر زمان ثبت تا زمان حرکت بررسی می‌شود. ویژگی LeadTimeDays می‌تواند یکی از بهترین سیگنال‌ها باشد، چون الگوی رزرو سفر کاری و تفریحی ممکن است متفاوت باشد (مثلاً سفر کاری نزدیک‌تر به حرکت رزرو شود یا برعکس). همچنین برای جلوگیری از خراب شدن نمودار با داده‌های پرت، مقادیر غیرمنطقی (کمتر از صفر) حذف می‌شوند و یک سقف پرت‌ها (مثلاً صدک ۹۹) اعمال می‌کنیم.

در این بخش، متغیر **LeadTimeDays** (فاصله زمانی بین ایجاد رزرو تا زمان حرکت) بین دو کلاس **Work** و **Int** مورد بررسی قرار گرفت. نتایج نمودار Boxplot نشان می‌دهد که میانه و بازه بین چارکی (IQR) در کلاس **Work** کمتر از کلاس **Int** است.

این یافته بیانگر آن است که سفرهای کاری معمولاً در فاصله زمانی کوتاه‌تری تا زمان حرکت رزرو می‌شوند، در حالی که سفرهای تفریحی به طور میانگین زودتر برنامه‌ریزی و ثبت می‌گردند و از تنوع رفتاری بیشتری برخوردارند.

وجود داده‌های پرت در هر دو کلاس نشان‌دهنده توزیع دم‌کلفت این متغیر است. همچنین با محدودسازی داده‌ها تا صدک ۹۹، از تأثیر مقادیر بسیار بزرگ بر تحلیل جلوگیری شده است.

در مجموع، تفاوت قابل مشاهده بین دو کلاس نشان می‌دهد که **LeadTimeDays** می‌تواند به عنوان یک ویژگی مؤثر در فرآیند مدل‌سازی و تفکیک اهداف سفر مورد استفاده قرار گیرد.

در این بخش بررسی می‌کنیم که برای هر نوع وسیله نقلیه (Vehicle)، سهم هر کلاس از TripReason چقدر است. برای این کار از جدول توافقی (Crosstab) با نرمال‌سازی سطری استفاده می‌کنیم تا خروجی به‌صورت درصد/نسبت باشد، نه تعداد خام. اگر برای بعضی وسیله‌ها سهم یکی از کلاس‌ها خیلی بیشتر باشد، این ستون می‌تواند یک ویژگی دسته‌ای بسیار مهم برای مدل باشد.

در این بخش، رابطه بین متغیر **Vehicle** و متغیر هدف **TripReason** با استفاده از جدول توافقی نرمال‌سازی‌شده (بر اساس سطرها) مورد بررسی قرار گرفت. در این نمایش، برای هر نوع وسیله نقلیه، سهم نسبی سفرهای کاری و تفریحی محاسبه شده است.

نتایج نشان می‌دهد که توزیع کلاس‌ها در میان انواع وسیله نقلیه متفاوت است. به‌عنوان مثال، در سفرهای اتوبوسی سهم سفرهای کاری بیشتر است، در حالی‌که در پروازهای بین‌المللی سهم سفرهای تفریحی غالب می‌باشد. سایر وسایل نقلیه نیز الگوهای متفاوتی از توزیع کلاس‌ها را نشان می‌دهند.

این تفاوت معنادار در نسبت‌ها بیانگر آن است که متغیر **Vehicle** می‌تواند یکی از ویژگی‌های مؤثر در پیش‌بینی دلیل سفر باشد. با این حال، برای تفسیر دقیق‌تر نتایج، بررسی تعداد نمونه‌های هر دسته نیز ضروری است تا از تأثیر احتمالی حجم کم داده در برخی گروه‌ها جلوگیری شود.

این بخش عمداً به‌عنوان یک نمونه EDA کم‌ارزش/نامناسب آورده می‌شود. ستون‌هایی مثل HashEmail (هش‌شده) معمولاً طوری ساخته می‌شوند که الگوی معنی‌دار برای پیش‌بینی ارائه نکنند و صرفاً نقش شناسایی/حریم خصوصی دارند. بنابراین بررسی طول هش (که غالباً ثابت یا نزدیک به ثابت است) هیچ بینش مفیدی برای مدل یا مسئله ایجاد نمی‌کند و حتی ممکن است ما را به تحلیل‌های بی‌اثر سوق دهد.

در این بخش، طول رشته‌ی متغیر **HashEmail** برای تمامی رکوردها بررسی شد. نتایج نشان می‌دهد که تقریباً تمامی مقادیر دارای طول ثابت (۵۶ کاراکتر) هستند و تنوع معناداری در این ویژگی مشاهده نمی‌شود.

ثابت بودن طول این متغیر نشان می‌دهد که ایمیل‌ها با یک الگوریتم هش استاندارد و با خروجی طول ثابت تولید شده‌اند. از آنجا که توابع هش به‌گونه‌ای طراحی می‌شوند که فاقد الگوهای قابل تفسیر باشند، این متغیر اطلاعات معناداری برای تفکیک کلاس‌های **TripReason** فراهم نمی‌کند.

در نتیجه، به دلیل نبود واریانس و عدم ارتباط تحلیلی با متغیر هدف، استفاده از **HashEmail** در فرآیند مدل‌سازی توصیه نمی‌شود و این ستون می‌تواند در مرحله پیش‌پردازش حذف گردد.

قبل از شروع مدل‌سازی، لازم است توزیع کلاس‌های متغیر هدف را بررسی کنیم. این کار به ما نشان می‌دهد آیا داده نامتوازن است یا خیر. اگر یک کلاس تعداد نمونه بیشتری داشته باشد، مدل ممکن است به سمت همان کلاس غالب متمایل شود و نتیجه‌گیری با معیارهایی مثل Accuracy گمراه‌کننده شود. به همین دلیل در این پروژه از معیار `F1-Score` استفاده می‌کنیم که تعادل بهتری بین Precision و Recall ایجاد می‌کند.

در این مرحله تعداد مقادیر یکتای هر ستون را بررسی می‌کنیم تا:
- ستون‌های با تنوع بسیار بالا (مثل شناسه‌ها یا مقادیر شبه‌هش) را شناسایی کنیم،
- تصمیم بگیریم کدام ستون‌ها ارزش یادگیری دارند و کدام ستون‌ها فقط نویز ایجاد می‌کنند،
- و در نهایت برای انتخاب روش مناسب کدگذاری (Encoding) و نحوه برخورد با ویژگی‌های دسته‌ای بهتر تصمیم بگیریم.

هر خرید با شناسه `BillID` ممکن است شامل چند بلیط باشد. تعداد بلیط‌ها در یک سفارش می‌تواند الگوی رفتاری سفر را نشان دهد؛ برای مثال سفرهای خانوادگی یا گروهی معمولاً چند بلیط دارند و ممکن است با تفریحی بودن سفر مرتبط باشند. بنابراین یک ویژگی جدید به نام `TicketPerOrder` می‌سازیم که تعداد بلیط‌های هر سفارش را نمایش می‌دهد.

در این بخش تلاش می‌کنیم سفارش‌هایی را شناسایی کنیم که در آن‌ها هم بلیط متعلق به مرد و هم بلیط متعلق به زن وجود دارد. ایده این است که وجود هر دو جنسیت در یک خرید

می‌تواند نشانه‌ای از سفر خانوادگی باشد. در نهایت یک ویژگی بولی (True/False) به نام family ایجاد می‌کنیم تا این الگو به شکل عددی/منطقی وارد مدل شود.

برخی ستون‌ها مانند UserID ، TicketID ، BillID و ستون‌های هش‌شده معمولاً برای پیش‌بینی علت سفر الگوی مفیدی ایجاد نمی‌کنند و بیشتر نقش شناسه دارند. این نوع ستون‌ها ممکن است باعث افزایش نویز، پیچیدگی و حتی overfitting شوند. بنابراین این ستون‌ها حذف می‌شوند تا مدل روی ویژگی‌های معنی‌دارتر تمرکز کند.

وجود مقدارهای گمشده (NaN) می‌تواند باعث خطا در مراحل بعدی (مانند Encoding یا مدل‌سازی) شود یا کیفیت مدل را کاهش دهد. در این مرحله تعداد مقدارهای گمشده هر ستون را بررسی می‌کنیم تا مشخص شود:
- کدام ستون‌ها نیاز به پر کردن دارند،
- و بهترین روش پر کردن برای هر ستون چیست.

در ستون VehicleClass تعدادی مقدار گمشده وجود دارد. برای جلوگیری از حذف تعداد زیادی از داده‌ها و همچنین جلوگیری از ایجاد خطا در مدل‌سازی، مقدارهای گمشده را با مقدار **مد** (بیشترین مقدار تکرارشونده) پر می‌کنیم. این روش معمولاً برای ویژگی‌های دسته‌ای/بولی انتخاب مناسبی است.

ستون‌های CancelTime ، Created و DepartureTime به صورت رشته هستند. برای اینکه بتوانیم ویژگی‌های زمانی استخراج کنیم (مثل اختلاف روزها یا ماه حرکت)، باید آن‌ها را به نوع داده‌ای datetime تبدیل کنیم. این مرحله پیش‌نیاز مهندسی ویژگی‌های زمانی است.

در این بخش اختلاف زمانی بین CancelTime و Created بررسی می‌شود تا درک بهتری از رفتار کنسلی کاربران داشته باشیم. این مرحله بیشتر برای شناخت داده انجام می‌شود و می‌تواند در تصمیم‌گیری‌های مهندسی ویژگی یا پاکسازی داده در مراحل بعدی مفید باشد.

اختلاف بین زمان حرکت (DepartureTime) و زمان ثبت بلیط (Created) می‌تواند یک سیگنال مهم برای تشخیص نوع سفر باشد. برای مثال ممکن است سفر کاری نزدیک‌تر به زمان حرکت رزرو شود یا برعکس. بنابراین ویژگی Departure_Created را به صورت اختلاف روزها استخراج می‌کنیم.

ماه حرکت می‌تواند نشان‌دهنده الگوهای فصلی و تعطیلات باشد. برای مثال در برخی ماه‌ها احتمال سفرهای تفریحی بیشتر است. به همین دلیل ماه حرکت را از DepartureTime

استخراج کرده و به عنوان یک ویژگی جدید به نام `DepartureMonth` اضافه می‌کنیم.

در بسیاری از ردیف‌ها مقدار `CancelTime` خالی است. برای جلوگیری از وجود مقادیر گمشده و ساده‌تر شدن پردازش، مقدارهای خالی `CancelTime` را با مقدار `Created` پر می‌کنیم. این کار باعث می‌شود در مراحل بعدی مشکلی برای مدل‌ها و تبدیل‌ها به وجود نیاید.

از آنجا که الگوریتم‌های یادگیری ماشین با برچسب‌های متنی به صورت مستقیم کار نمی‌کنند، مقادیر متنی ستون `TripReason` (مثل Work و Int) را به اعداد تبدیل می‌کنیم. همچنین نگاشت کلاس‌ها چاپ می‌شود تا مشخص باشد هر عدد نماینده کدام کلاس است.

ستون `Vehicle` یک ویژگی دسته‌ای بدون ترتیب است (Plane/Bus/Train و ...). برای اینکه مدل بتواند هر دسته را به شکل مستقل یاد بگیرد، از روش **One-Hot Encoding** استفاده می‌کنیم. در این روش برای هر نوع Vehicle یک ستون جدا ساخته می‌شود و مقدار ۱ یا ۰ نشان‌دهنده وجود یا عدم وجود آن دسته است.

پس از استخراج ویژگی‌های زمانی مورد نیاز (مانند `Departure_Created` و `DepartureMonth`)، ستون‌های خام زمانی (`DepartureTime` ، `CancelTime` ، `Created`) حذف می‌شوند، چون اطلاعات مفیدشان به شکل عددی استخراج شده و نگه داشتن آن‌ها معمولاً کمکی به مدل‌های کلاسیک نمی‌کند.

پس از حذف برخی ستون‌ها، هنوز چند ویژگی دسته‌ای (مثل مبدا/مقصد و وضعیت رزرو) باقی می‌مانند که باید به عدد تبدیل شوند. در اینجا از `OrdinalEncoder` استفاده می‌کنیم که به هر دسته یک عدد اختصاص می‌دهد. این روش ساده و سریع است و برای مدل‌هایی مثل درخت تصمیم، رندوم‌فارست و XGBoost معمولاً نتیجه قابل قبولی می‌دهد.

ستون `BuyerMobile` (هش‌شده) معمولاً تنوع بسیار بالایی دارد و الگوی قابل اتکایی برای پیش‌بینی علت سفر ارائه نمی‌کند. وجود چنین ستون‌هایی ممکن است باعث افزایش نویز و پیچیده شدن داده شود. بنابراین این ستون حذف می‌شود تا مدل روی ویژگی‌های مؤثرتر تمرکز کند.

وجود مقادیر منفی برای `CouponDiscount` معمولاً نشان‌دهنده داده‌های نامعتبر یا خطای ثبت است. بنابراین ردیف‌های دارای تخفیف منفی حذف می‌شوند. سپس برای نزدیک شدن به مبلغ واقعی، قیمت نهایی را محاسبه می‌کنیم:

```
Price = Price - CouponDiscount
```

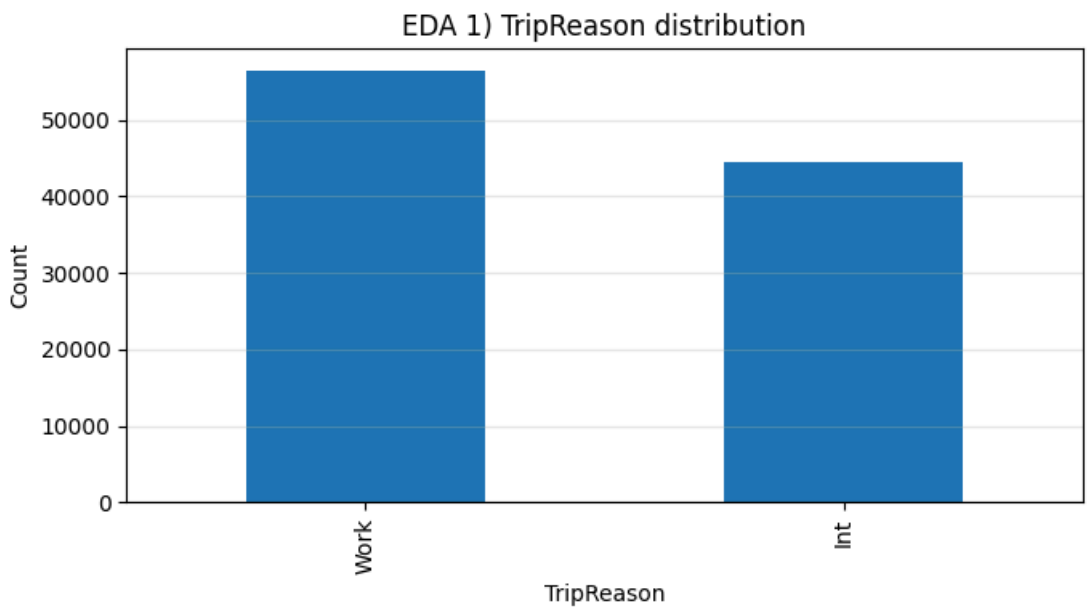
وجود تخفیف می‌تواند نشانه‌ای از رفتار خرید باشد؛ برای مثال ممکن است سفرهای تفریحی بیشتر با تخفیف خریداری شوند. به همین دلیل یک ویژگی بولی به نام `Discount` ایجاد می‌کنیم که نشان دهد آیا تخفیف وجود دارد یا خیر. سپس ستون `CouponDiscount` حذف می‌شود چون اثر آن در قیمت و ویژگی Discount لحاظ شده است.

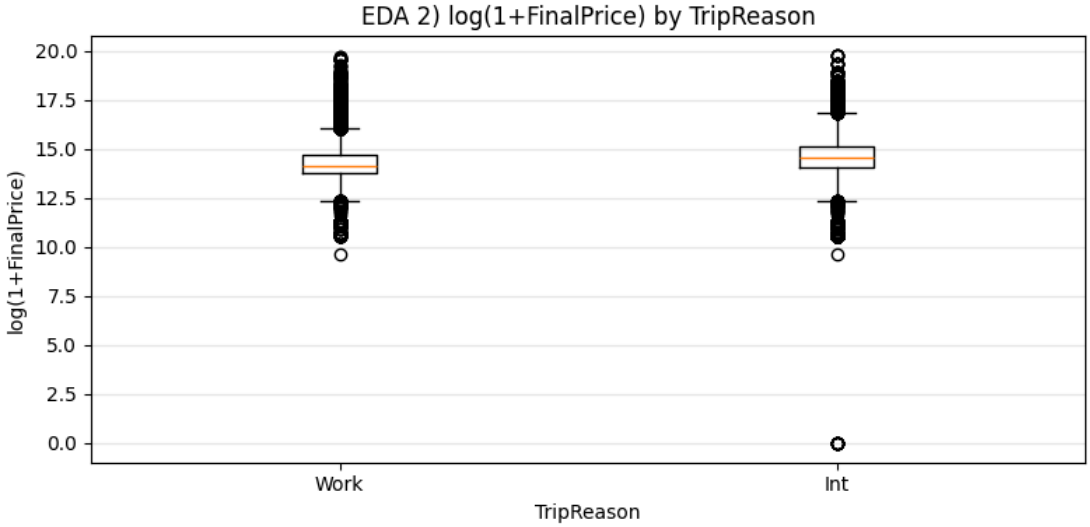
پس از اعمال تخفیف ممکن است برخی قیمت‌ها منفی یا صفر شوند که از نظر منطقی برای بلیط معتبر نیستند. در داده تست قیمت‌های منفی اصلاح می‌شوند تا مقدار عددی قابل استفاده داشته باشند. در داده آموزش نیز ردیف‌هایی با `Price <= 0` حذف می‌شوند تا مدل از داده‌های غیرواقعی یاد نگیرد.

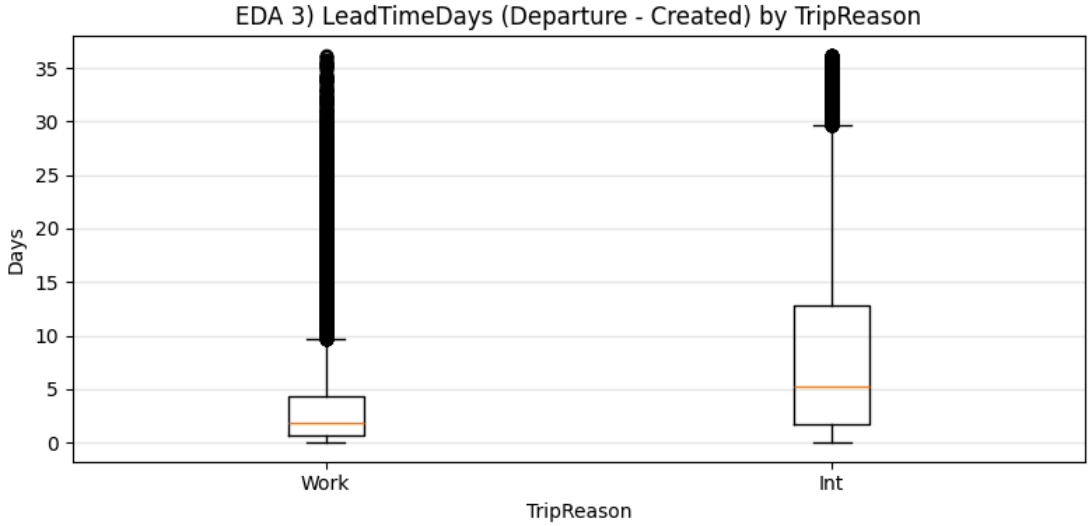
برای کاهش اثر قیمت‌های بسیار غیرعادی (پرت‌ها) از روش IQR استفاده می‌کنیم. ابتدا چارک اول و سوم محاسبه می‌شود و سپس بازه قابل قبول تعریف می‌گردد. با حذف ردیف‌هایی که قیمت آن‌ها خارج از این بازه است، مدل پایدارتر می‌شود و حساسیت آن نسبت به داده‌های غیرواقعی کمتر خواهد شد.

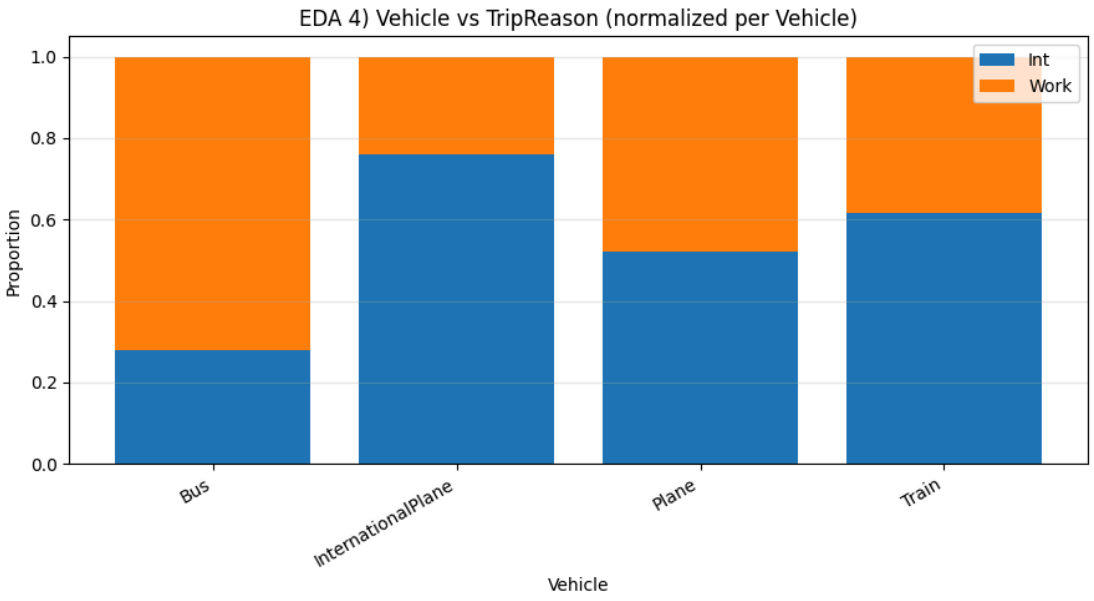
نرمال‌سازی (Scaling) همیشه اجباری نیست و به نوع مدل بستگی دارد. برخی مدل‌ها به مقیاس ویژگی‌ها حساس‌اند (مثل **Logistic Regression** ، **KNN** و **SVM**)؛ چون بر اساس فاصله یا بهینه‌سازی عددی عمل می‌کنند و اگر یک ویژگی (مثل Price) دامنه خیلی بزرگی داشته باشد می‌تواند روی تصمیم مدل اثر نامتعادل بگذارد. در مقابل، مدل‌های مبتنی بر درخت (مثل **Random Forest** ، **Decision Tree** و **XGBoost**) معمولاً به مقیاس ویژگی‌ها حساس نیستند و بدون نرمال‌سازی هم عملکرد خوبی دارند.
بنابراین در این پروژه نرمال‌سازی را به عنوان یک مرحله **اختیاری** در نظر می‌گیریم و فقط زمانی انجام می‌دهیم که بخواهیم مدل‌های حساس به مقیاس را آموزش دهیم.

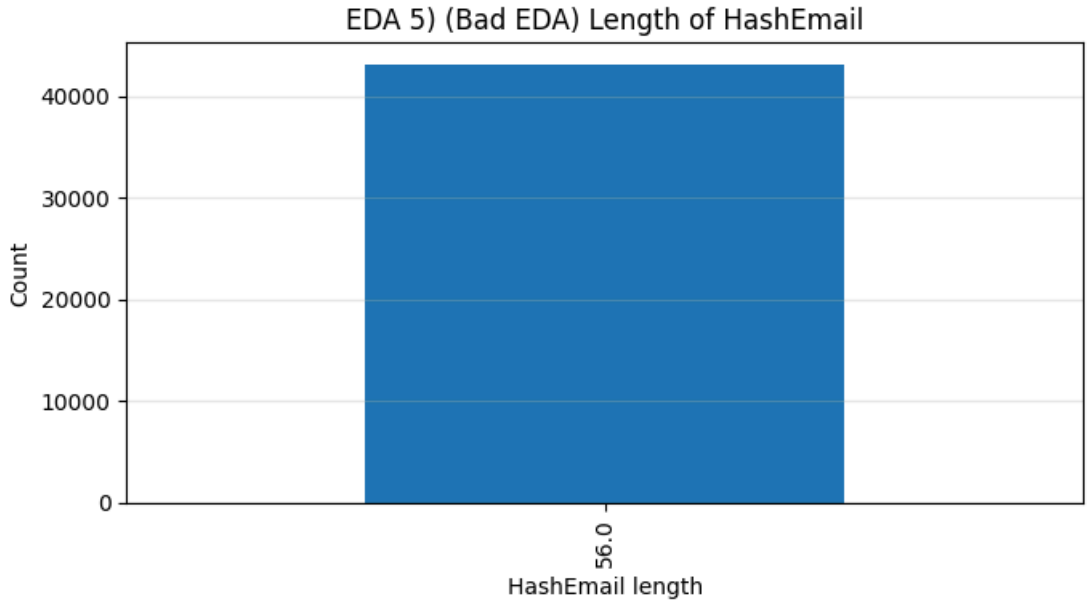
ذخیره سازی دیتا


EDA 1) TripReason distribution
Count
50000
40000
30000
20000
10000
0
Work
Int
TripReason


EDA 2) log(1+FinalPrice) by TripReason
log(1+FinalPrice)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Work
Int
TripReason


EDA 3) LeadTimeDays (Departure - Created) by TripReason
Days
35
30
25
20
15
10
5
0
Work
Int
TripReason


EDA 4) Vehicle vs TripReason (normalized per Vehicle)
Int
Work
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
Bus
InternationalPlane
Plane
Train
Vehicle


EDA 5) (Bad EDA) Length of HashEmail
Count
40000
30000
20000
10000
0
56.0
HashEmail length

TicketID	BillID	DepartureTime	CancelTime	Created	
1091777.0	39710203	2022-11-02 23:59:00	NaN	2022-10-23 09:38:49.110	**0**
1070902.0	38689463	2022-08-18 04:15:00	NaN	2022-08-15 14:51:43.160	**1**
7624237.0	39245173	2022-09-21 11:00:00	NaN	2022-09-20 17:25:27.250	**2**
2867547.0	37957585	2022-06-26 08:30:00	NaN	2022-06-25 11:32:53.980	**3**
7212559.0	37584530	2022-06-02 23:00:00	NaN	2022-06-01 11:30:53.633	**4**
...	...	...	...	...	**...**
1050781.0	37579327	2022-06-04 12:10:00	NaN	2022-06-01 00:20:14.280	**101012**
3085407.0	39789479	2022-11-01 15:30:00	NaN	2022-10-29 20:54:31.330	**101013**
2322052.0	38991563	2022-09-13 09:30:00	NaN	2022-09-03 17:57:22.067	**101014**
7664730.0	39406503	2022-09-29 17:30:00	NaN	2022-09-29 13:15:51.303	**101015**
2711094.0	36742809	2022-04-05 17:10:00	NaN	2022-03-31 13:29:11.530	**101016**

2. رگرسیون لجستیک (Logistic Regression)

در این نوتبوک هدف ما آموزش و تنظیم (Tuning) مدل **Logistic Regression** است. لجستیک رگرشن یک مدل خطی برای طبقه‌بندی است که به عنوان یک مدل پایه (Baseline) بسیار رایج و قابل دفاع محسوب می‌شود.

نکته مهم این است که Logistic Regression نسبت به مقیاس ویژگی‌ها حساس است؛ بنابراین در این نوتبوک از **Scaling** (مانند `MinMaxScaler` یا StandardScaler) داخل `Pipeline` استفاده می‌کنیم تا:

- ویژگی‌ها هم‌مقیاس شوند،
- و در Cross-Validation نشت اطلاعات (Data Leakage) رخ ندهد.

در نوتبوک پیش‌پردازش، داده‌ها پاکسازی و به فرم عددی مناسب برای مدل‌سازی تبدیل شده‌اند. در این نوتبوک فقط روی مدل Logistic Regression تمرکز می‌کنیم؛ بنابراین داده آماده را از فایل می‌خوانیم تا ساختار پروژه مرتب و قابل توسعه باشد.

برای آموزش مدل باید ستون هدف `TripReason` را به عنوان y جدا کنیم و سایر ستون‌ها را به عنوان ویژگی‌های ورودی X در نظر بگیریم. این کار باعث می‌شود ورودی مدل دقیقاً شامل ویژگی‌ها باشد و از بروز خطای مفهومی جلوگیری شود.

برای ارزیابی واقعی مدل، بخشی از داده را به عنوان Validation کنار می‌گذاریم. همچنین با استفاده از `stratify` نسبت کلاس‌ها در Train و Validation مشابه می‌ماند و ارزیابی منصفانه‌تر می‌شود.

Logistic Regression یک مدل خطی است و در فرآیند یادگیری به مقیاس ویژگی‌ها حساس است؛ اگر یک ویژگی دامنه بسیار بزرگ‌تری داشته باشد، می‌تواند روی یادگیری اثر غالب بگذارد.

به همین دلیل از Scaling استفاده می‌کنیم. همچنین Scaling را داخل `Pipeline` قرار می‌دهیم تا در Cross-Validation، scaler فقط روی داده Train هر fold fit شود و نشت اطلاعات (Data Leakage) رخ ندهد.

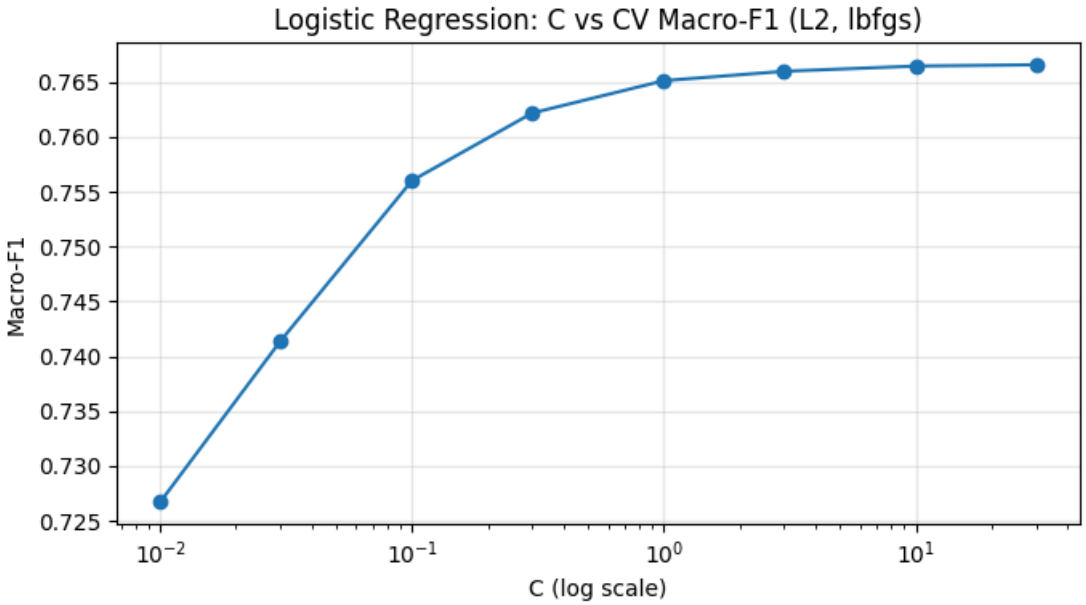
برای ارزیابی مدل از `F1-Score` استفاده می‌کنیم چون معیار داوری همین است. در این نوتبوک از **Macro-F1** استفاده می‌کنیم تا هر کلاس وزن برابر داشته باشد و در صورت نامتوازن بودن داده، مدل فقط روی کلاس غالب خوب عمل نکند.

برای اینکه تست ما واقعی باشد، فقط یک بار مدل را آموزش نمی‌دهیم. پارامترهای مهم Logistic Regression را بررسی می‌کنیم:

- `C` : میزان منظم‌سازی (Regularization). C کوچک‌تر یعنی منظم‌سازی بیشتر (مدل ساده‌تر)، و C بزرگ‌تر یعنی منظم‌سازی کمتر (مدل پیچیده‌تر).
- `penalty` : نوع منظم‌سازی (L1 یا L2)
- `solver` : الگوریتم بهینه‌سازی که باید با penalty سازگار باشد.

برای انتخاب بهترین ترکیب، از Cross-Validation استفاده می‌کنیم تا نتیجه به یک تقسیم‌بندی خاص وابسته نباشد.

پس از انتخاب بهترین پارامترها، مدل نهایی را با همان تنظیمات آموزش می‌دهیم و روی داده Validation ارزیابی می‌کنیم. در کنار F1-Score، ماتریس درهم‌ریختگی و گزارش طبقه‌بندی را نیز نمایش می‌دهیم تا رفتار مدل در هر کلاس مشخص شود.



hicleClass	Domestic	To	From	Price	Male	ReserveStatus	
0.0	1	80.0	217.0	6565575.0	True	5	0
0.0	1	80.0	171.0	9500000.0	True	5	1
1.0	1	37.0	135.0	2000000.0	False	3	2
1.0	1	85.0	64.0	40000.0	False	2	3
1.0	1	244.0	68.0	1130000.0	True	3	4

cv_std_f1	cv_mean_f1	solver	penalty	C	
0.002922	0.766524	liblinear	l1	1.00	10
0.002911	0.766452	lbfgs	l2	10.00	3
0.002856	0.766415	liblinear	l1	10.00	11
0.002972	0.766340	liblinear	l2	10.00	7
0.003132	0.765708	liblinear	l1	0.10	9
0.002950	0.765122	lbfgs	l2	1.00	2
0.002938	0.765008	liblinear	l2	1.00	6
0.003761	0.756015	lbfgs	l2	0.10	1
0.003887	0.755836	liblinear	l2	0.10	5
0.003828	0.754902	liblinear	l1	0.01	8

3. الگوریتم K-نزدیک‌ترین همسایه (K-Nearest Neighbors)

در این نوتبوک هدف ما آموزش و تنظیم (Tuning) مدل **KNN** است. برای این کار به کتابخانه‌هایی نیاز داریم که:

- داده‌ها را بخوانیم و مدیریت کنیم (`pandas` ، `numpy`)
- نمودار تحلیل نتایج رسم کنیم (`matplotlib`)
- داده را به بخش آموزش و اعتبارسنجی تقسیم کنیم (`train_test_split`)
- مدل KNN را بسازیم و پارامترهای آن را آزمایش کنیم (`KNeighborsClassifier`)
- و از معیار `F1-Score` برای ارزیابی استفاده کنیم.

در نوتبوک قبلی (پیش‌پردازش)، داده‌ها پاکسازی و مهندسی ویژگی روی آن انجام شده است و خروجی آن به صورت فایل ذخیره شده. در این نوتبوک فقط روی **مدل‌سازی** تمرکز می‌کنیم؛ بنابراین داده‌های آماده‌شده را از فایل می‌خوانیم تا:

- مراحل پروژه ما ماژولار و مرتب باشد،
- تکرار بی‌مورد کدها کم شود،
- و بتوانیم برای هر مدل یک نوتبوک جدا و تمیز داشته باشیم.

برای آموزش هر مدل یادگیری ماشین، باید:

- ستون هدف (`TripReason`) را به عنوان **y** جدا کنیم،
- و بقیه ستون‌ها را به عنوان ویژگی‌ها (**X**) استفاده کنیم.

این جداسازی باعث می‌شود فرآیند مدل‌سازی شفاف باشد و اشتباهاً ستون هدف وارد ورودی مدل نشود.

برای اینکه بتوانیم عملکرد مدل را به صورت واقعی ارزیابی کنیم، بخشی از داده را برای **اعتبارسنجی (Validation)** کنار می‌گذاریم. مدل فقط روی داده Train آموزش می‌بیند و سپس روی Validation تست می‌شود.

همچنین از `stratify` استفاده می‌کنیم تا نسبت کلاس‌های `TripReason` در Train و Validation مشابه باشد؛ این کار به خصوص زمانی مهم است که داده نامتوازن باشد.

الگوریتم **KNN** بر اساس **فاصله** بین نمونه‌ها تصمیم می‌گیرد. بنابراین اگر یکی از ویژگی‌ها (مثل Price) دامنه بسیار بزرگ‌تری نسبت به سایر ویژگی‌ها داشته باشد، روی فاصله‌ها اثر غالب می‌گذارد و مدل منحرف می‌شود.

به همین دلیل قبل از KNN معمولاً از **Scaling** استفاده می‌شود (مثل `MinMaxScaler`).

همچنین Scaling را داخل `Pipeline` قرار می‌دهیم تا:
- عملیات Scaling فقط روی Train fit شود (جلوگیری از نشت اطلاعات یا Data Leakage)،
- و در Cross-Validation به شکل کاملاً صحیح و خودکار انجام شود.

چون معیار داوری این تمرین `F1-Score` است، ما هم مدل‌ها را بر اساس همین معیار ارزیابی می‌کنیم. در این نوتبوک از **Macro-F1** استفاده می‌کنیم تا هر کلاس وزن برابر داشته باشد و اگر داده نامتوازن باشد، مدل فقط روی کلاس غالب خوب عمل نکند.

پارامتر اصلی در KNN مقدار `k` (تعداد همسایه‌ها) است:
- k کوچک → مدل حساس‌تر و ممکن است overfit شود،
- k بزرگ → مدل نرم‌تر می‌شود ولی ممکن است underfit کند.

برای انتخاب k مناسب، به جای یک تست ساده، از **Cross-Validation** استفاده می‌کنیم تا نتیجه به یک تقسیم‌بندی خاص وابسته نباشد. این روش باعث می‌شود ارزیابی پایدارتر و قابل اعتمادتر باشد.

بعد از محاسبه امتیاز برای k های مختلف، یک نمودار رسم می‌کنیم تا:
- روند تغییر عملکرد مدل نسبت به k را ببینیم،
- نقطه بهینه را بهتر تشخیص دهیم،
- و تصمیم انتخاب k را قابل دفاع‌تر کنیم (برای گزارش و فاز ارائه).

پس از انتخاب بهترین پارامترها با Cross-Validation، مدل نهایی را با همان تنظیمات آموزش می‌دهیم و سپس روی داده Validation ارزیابی می‌کنیم.

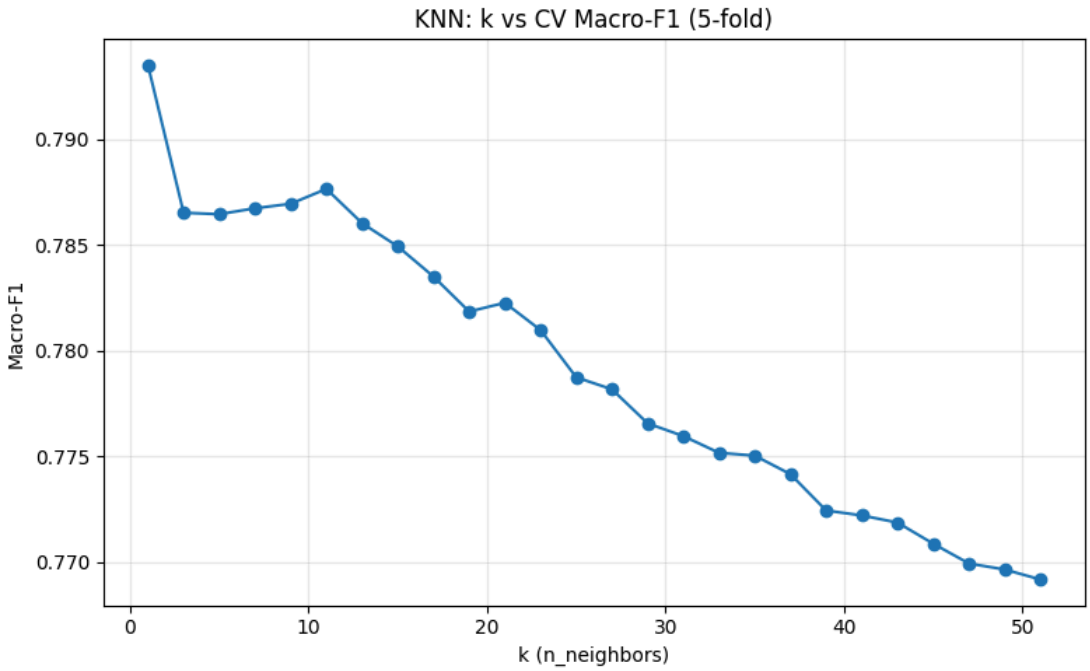
در این مرحله علاوه بر F1-Score، خروجی‌های زیر را هم نمایش می‌دهیم:
- `Confusion Matrix` برای بررسی نوع خطاها

- `Classification Report` برای Precision/Recall هر کلاس

این گزارش‌ها کمک می‌کنند نقاط ضعف مدل را دقیق‌تر تحلیل کنیم.

بعد از اینکه مدل را ارزیابی کردیم و از تنظیمات آن مطمئن شدیم، برای بیشترین استفاده از داده، مدل را روی کل داده آموزشی (Train + Validation) آموزش می‌دهیم.

سپس با این مدل، مقادیر `TripReason` را برای داده تست پیش‌بینی می‌کنیم و دیتافریم `submission` را می‌سازیم. در نهایت فایل `submission.csv` ذخیره می‌شود تا در سامانه داوری بارگذاری شود.



hicleClass	Domestic	To	From	Price	Male	ReserveStatus	
0.0	1	80.0	217.0	6565575.0	True	5	0
0.0	1	80.0	171.0	9500000.0	True	5	1
1.0	1	37.0	135.0	2000000.0	False	3	2
1.0	1	85.0	64.0	40000.0	False	2	3
1.0	1	244.0	68.0	1130000.0	True	3	4

cv_std_f1	cv_mean_f1	p	weights	k	
0.001723	0.827431	1	distance	29	54
0.001476	0.827236	1	distance	39	59
0.001789	0.827214	1	distance	35	57
0.001459	0.827195	1	distance	37	58
0.001964	0.827170	1	distance	27	53
0.001861	0.826986	1	distance	31	55
0.001662	0.826957	1	distance	25	52
0.001651	0.826948	1	distance	33	56
0.001814	0.826831	1	distance	23	51
0.001807	0.826799	1	distance	21	50

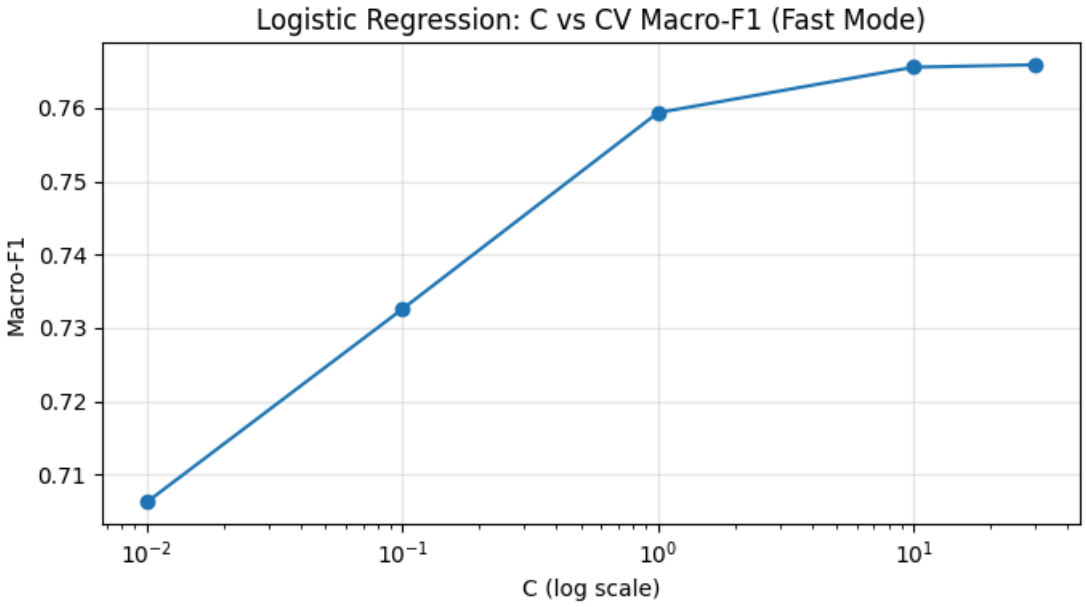
4. ماشین بردار پشتیبان (Support Vector Machine)

برخی مدل‌ها مانند **SVM** به‌خصوص با کرنل‌های غیرخطی (مثل RBF) زمان آموزش بالایی دارند. برای اینکه فرآیند تست و تنظیم پارامترها سریع‌تر انجام شود، در این نوتبوک فقط **۲۰٪** از داده‌های آموزشی را استفاده می‌کنیم.

برای اینکه نسبت کلاس‌ها به‌هم نریزد، نمونه‌گیری را به صورت **Stratified** انجام می‌دهیم تا توزیع `TripReason` در این ۲۰٪ مشابه کل داده باقی بماند.

SVM یکی از مدل‌های قدرتمند برای طبقه‌بندی است و با انتخاب کرنل مناسب می‌تواند مرزهای تصمیم پیچیده را یاد بگیرد. اما SVM معمولاً به مقیاس ویژگی‌ها حساس است؛ بنابراین برای عملکرد بهتر از **Scaling** استفاده می‌کنیم.

همچنین چون SVM می‌تواند زمان‌بر باشد، برای تست‌های اولیه از نمونه‌گیری ۲۰٪ و از Cross-Validation با تعداد fold کمتر استفاده می‌کنیم تا زمان اجرا منطقی بماند.



hicleClass	Domestic	To	From	Price	Male	ReserveStatus	
0.0	1	80.0	217.0	6565575.0	True	5	0
0.0	1	80.0	171.0	9500000.0	True	5	1
1.0	1	37.0	135.0	2000000.0	False	3	2
1.0	1	85.0	64.0	40000.0	False	2	3
1.0	1	244.0	68.0	1130000.0	True	3	4

cv_std_f1	cv_mean_f1	gamma	C	kernel	
0.003617	0.766798	scale	10.0	rbf	10
0.003239	0.759472	0.1	10.0	rbf	11
0.006195	0.753485	None	10.0	linear	2
0.006951	0.752198	None	30.0	linear	3
0.005196	0.746373	None	1.0	linear	1
0.002731	0.743635	scale	1.0	rbf	7
0.002863	0.729914	0.1	1.0	rbf	8
0.004815	0.728152	0.01	10.0	rbf	12
0.002037	0.711531	scale	0.1	rbf	4
0.005810	0.701826	None	0.1	linear	0

5. جنگل تصادفی (Random Forest)

در این نوتبوک هدف ما آموزش و تنظیم (Tuning) مدل **Random Forest** است. رندوم‌فارست یک مدل مبتنی بر درخت است که با ساخت چندین درخت و تجمیع نتایج آن‌ها، معمولاً دقت و پایداری بالایی به دست می‌دهد.

مزیت مهم Random Forest این است که برخلاف مدل‌های مبتنی بر فاصله (مثل KNN)، معمولاً به نرمال‌سازی ویژگی‌ها حساس نیست و بدون Scaling هم عملکرد خوبی دارد.

در نوتبوک پیش‌پردازش، داده‌ها پاکسازی و مهندسی ویژگی شده و در فایل ذخیره شده‌اند. در این نوتبوک تمرکز ما فقط روی مدل‌سازی Random Forest است؛ بنابراین داده‌های آماده را از فایل می‌خوانیم تا ساختار پروژه مرتب و ماژولار باشد.

برای آموزش مدل، ستون هدف `TripReason` را به عنوان **y** جدا می‌کنیم و باقی ستون‌ها را به عنوان ویژگی‌های ورودی **X** در نظر می‌گیریم. این مرحله از وارد شدن ناخواسته ستون هدف به ورودی مدل جلوگیری می‌کند و ساختار مدل‌سازی را شفاف می‌سازد.

برای ارزیابی واقعی عملکرد مدل، بخشی از داده را به عنوان مجموعه اعتبارسنجی کنار می‌گذاریم. همچنین از `stratify` استفاده می‌کنیم تا نسبت کلاس‌های TripReason در هر دو بخش مشابه باقی بماند و ارزیابی منصفانه‌تر شود.

Random Forest یک مدل مبتنی بر درخت تصمیم است و تصمیم‌گیری آن بر اساس **آستانه‌گذاری (split)** روی ویژگی‌ها انجام می‌شود، نه بر اساس فاصله بین نمونه‌ها. به همین دلیل معمولاً به مقیاس ویژگی‌ها حساس نیست و نیازی به نرمال‌سازی/استانداردسازی ندارد. این موضوع باعث می‌شود مدل‌سازی با Random Forest ساده‌تر باشد و مراحل Scaling را حذف کنیم.

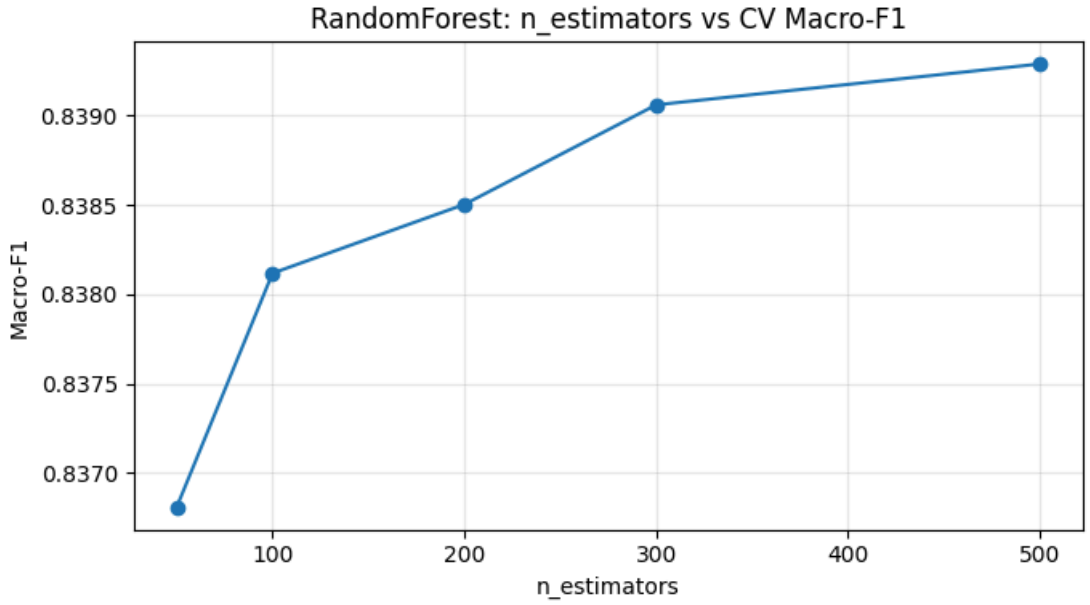
با توجه به اینکه معیار داوری این تمرین `F1-Score` است، ما نیز مدل را با همین معیار می‌سنجیم. از **Macro-F1** استفاده می‌کنیم تا هر کلاس وزن برابر داشته باشد و مدل فقط روی کلاس غالب خوب عمل نکند.

برای اینکه به بهترین عملکرد برسیم، فقط یک بار مدل را آموزش نمی‌دهیم؛ بلکه پارامترهای مهم Random Forest را تست می‌کنیم، مثل:

- `n_estimators` : تعداد درخت‌ها (معمولاً هرچه بیشتر، پایدارتر ولی کندتر)
- `max_depth` : عمق درخت‌ها (کنترل overfitting/underfitting)
- `min_samples_split` و `min_samples_leaf` : جلوگیری از ساخت برگ‌های خیلی کوچک
- `max_features` : تعداد ویژگی‌هایی که هر درخت در هر split بررسی می‌کند

برای ارزیابی منصفانه و پایدار از **Cross-Validation** استفاده می‌کنیم.

پس از پیدا کردن بهترین پارامترها، مدل نهایی را با همان تنظیمات آموزش می‌دهیم و روی داده Validation ارزیابی می‌کنیم. علاوه بر F1-Score، ماتریس درهم‌ریختگی و گزارش طبقه‌بندی را نیز نمایش می‌دهیم تا نوع خطاهای مدل مشخص شود.



hicleClass	Domestic	To	From	Price	Male	ReserveStatus	
0.0	1	80.0	217.0	6565575.0	True	5	0
0.0	1	80.0	171.0	9500000.0	True	5	1
1.0	1	37.0	135.0	2000000.0	False	3	2
1.0	1	85.0	64.0	40000.0	False	2	3
1.0	1	244.0	68.0	1130000.0	True	3	4

cv_std_f1	cv_mean_f1	min_samples_split	min_samples_leaf	
0.002595	0.844093	2	1	0
0.002105	0.841037	5	1	1
0.003065	0.836753	2	2	4
0.002388	0.836191	5	2	5
0.002454	0.835127	10	1	2
0.003047	0.831721	10	2	6
0.003259	0.826666	20	1	3
0.002489	0.823878	20	2	7
0.002758	0.823244	2	5	8
0.002758	0.823244	5	5	9

6. XGBoost

در این نوتبوک هدف ما آموزش و تنظیم (Tuning) مدل **XGBoost** است. XGBoost یک الگوریتم قدرتمند مبتنی بر درخت‌های تصمیم و روش **Boosting** است که معمولاً در مسائل جدولی (Tabular Data) عملکرد بسیار خوبی دارد.

برخلاف مدل‌هایی مثل KNN، XGBoost معمولاً به Scaling حساس نیست؛ چون تصمیم‌گیری آن بر اساس تقسیم‌بندی درختی انجام می‌شود نه فاصله.

در ادامه، علاوه بر آموزش مدل، چندین پارامتر مهم را تست می‌کنیم تا بهترین تنظیمات ممکن بر اساس معیار `F1-Score` به دست آید.

در نوتبوک پیش‌پردازش، داده‌ها پاکسازی و تبدیل به فرم عددی شده‌اند. در این نوتبوک تمرکز فقط روی مدل‌سازی است؛ بنابراین داده‌های آماده را از فایل می‌خوانیم تا پروژه ما:
- مرتب و ماژولار باشد،
- و بتوانیم برای هر مدل یک نوتبوک مستقل داشته باشیم.

برای مدل‌سازی باید ستون هدف `TripReason` را به عنوان y جدا کنیم و باقی ستون‌ها را به عنوان ویژگی‌های ورودی X در نظر بگیریم. این کار از ورود ناخواسته ستون هدف به ورودی مدل جلوگیری کرده و روند آموزش را شفاف می‌کند.

برای ارزیابی واقعی مدل، بخشی از داده را به عنوان Validation کنار می‌گذاریم. با استفاده از `stratify` نسبت کلاس‌ها در هر دو بخش مشابه می‌ماند تا ارزیابی منصفانه‌تر و پایدارتر شود.

چون معیار داوری مسئله `F1-Score` است، ما نیز مدل را با همین معیار ارزیابی می‌کنیم. از **Macro-F1** استفاده می‌کنیم تا هر کلاس وزن برابر داشته باشد و مدل فقط روی کلاس غالب خوب عمل نکند.

در ابتدا یک مدل پایه با تنظیمات معقول آموزش می‌دهیم تا یک نقطه شروع داشته باشیم. سپس در مراحل بعد با تنظیم پارامترها (Tuning) تلاش می‌کنیم عملکرد را بهبود دهیم.

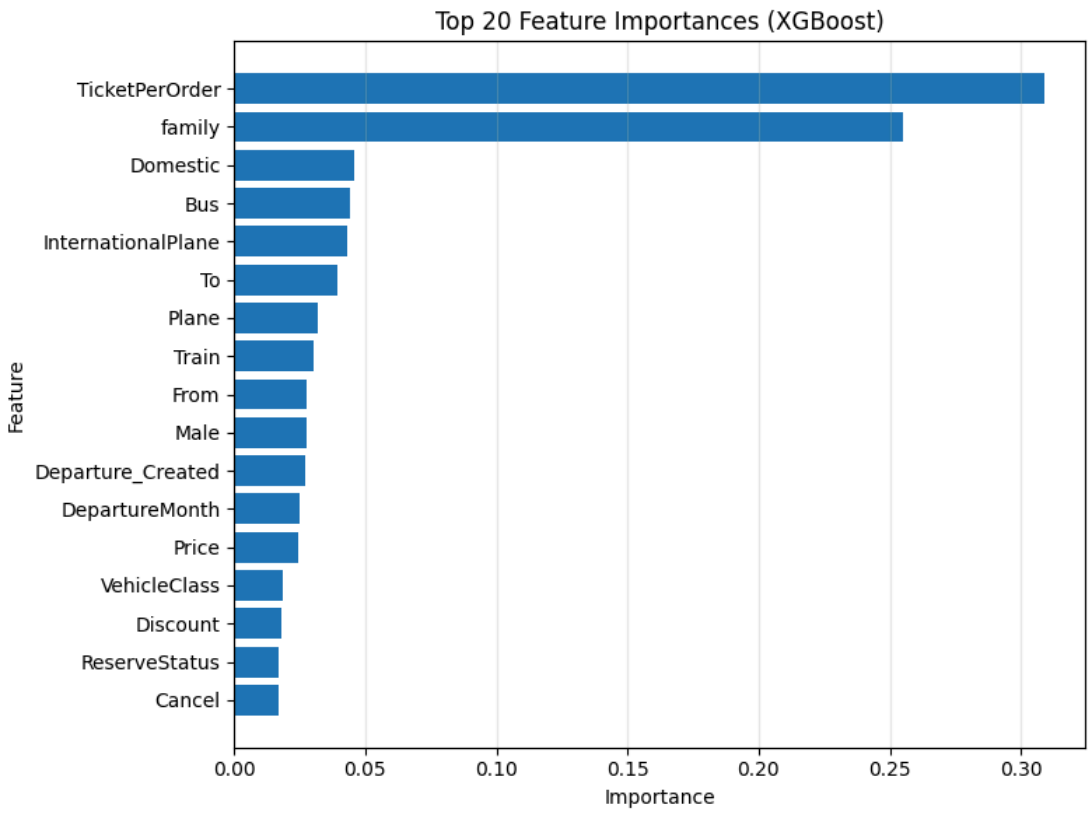
XGBoost پارامترهای زیادی دارد و انتخاب دستی بهترین ترکیب همیشه ساده نیست. برای اینکه تست ما واقعی و قابل دفاع باشد، از **Cross-Validation** به همراه **RandomizedSearch** استفاده می‌کنیم تا چندین ترکیب از پارامترها بررسی شود.

برخی پارامترهای مهم:

- `n_estimators` : تعداد درخت‌ها
- `max_depth` : عمق هر درخت (کنترل پیچیدگی)
- `learning_rate` : نرخ یادگیری (Trade-off با n_estimators)
- `subsample` و `colsample_bytree` : جلوگیری از overfitting
- `min_child_weight` و `gamma` : کنترل سخت‌گیری برای split ها

پس از یافتن بهترین پارامترها، مدل نهایی را با همان تنظیمات روی داده Train آموزش می‌دهیم و روی Validation ارزیابی می‌کنیم. علاوه بر F1-Score، ماتریس درهم‌ریختگی و گزارش طبقه‌بندی نمایش داده می‌شود تا نوع خطاهای مدل مشخص شود.

یکی از مزیت‌های مدل‌های مبتنی بر درخت امکان بررسی اهمیت ویژگی‌هاست. در این بخش اهمیت ویژگی‌ها را استخراج می‌کنیم تا بفهمیم کدام ویژگی‌ها بیشترین نقش را در پیش‌بینی TripReason داشته‌اند.



hicleClass	Domestic	To	From	Price	Male	ReserveStatus	
0.0	1	80.0	217.0	6565575.0	True	5	0
0.0	1	80.0	171.0	9500000.0	True	5	1
1.0	1	37.0	135.0	2000000.0	False	3	2
1.0	1	85.0	64.0	40000.0	False	2	3
1.0	1	244.0	68.0	1130000.0	True	3	4

importance	feature	
0.309011	TicketPerOrder	8
0.255024	family	9
0.045729	Domestic	5
0.044282	Bus	12
0.042906	InternationalPlane	13
0.039208	To	4
0.031833	Plane	14
0.030152	Train	15
0.027755	From	3
0.027475	Male	1
0.026973	Departure_Created	10
0.024838	DepartureMonth	11
0.024552	Price	2
0.018699	VehicleClass	6
0.017829	Discount	16
0.016904	ReserveStatus	0
0.016831	Cancel	7

7. منحنی‌های ROC و مقایسه مدل‌ها

در این نوتبوک، منحنی **ROC (Receiver Operating Characteristic)** را برای تمام مدل‌های آموزش‌دیده رسم می‌کنیم.

منحنی ROC نشان‌دهنده Trade-off بین **True Positive Rate** و **False Positive Rate** در آستانه‌های مختلف تصمیم‌گیری است.

معیار **AUC (Area Under Curve)** نشان‌دهنده توانایی کلی مدل در تفکیک کلاس‌ها است؛ هرچه به ۱ نزدیک‌تر باشد، مدل بهتر عمل می‌کند.

از همان داده پیش‌پردازش‌شده و همان `random_state=42` استفاده می‌کنیم تا تقسیم‌بندی Train/Validation دقیقاً مشابه سایر نوتبوک‌ها باشد و نتایج قابل مقایسه باشند.

هر مدل با بهترین پارامترهایی که در نوتبوک مربوطه پیدا شده، ساخته می‌شود.

نکته مهم: برای SVM از `probability=True` استفاده می‌کنیم تا بتوانیم احتمال پیش‌بینی را به دست آوریم.

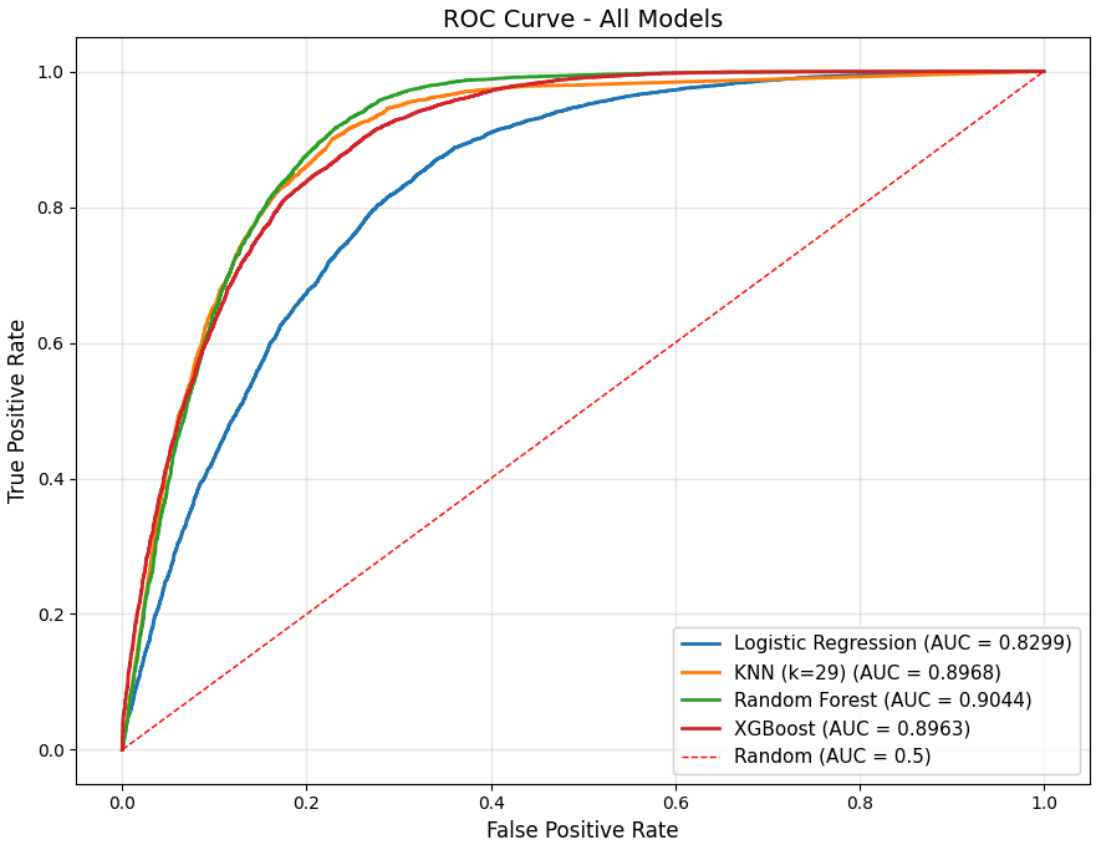
برای رسم منحنی ROC به احتمال تعلق هر نمونه به کلاس مثبت (`predict_proba`) نیاز داریم.

هر مدل روی داده Train آموزش می‌بیند و سپس احتمال پیش‌بینی روی Validation محاسبه می‌شود.

تمام منحنی‌ها در یک نمودار رسم می‌شوند تا مقایسه دیداری آسان باشد.

خط چین قرمز نشان‌دهنده مدل تصادفی (Random Classifier) با AUC=0.5 است.

برای مقایسه عددی، مقدار AUC هر مدل در یک جدول نمایش داده می‌شود.



AUC	Model	
0.9044	Random Forest	0
0.8968	KNN (k=29)	1
0.8963	XGBoost	2
0.8299	Logistic Regression	3

8. نتایج نهایی و جمع‌بندی

در این پژوهش، عملکرد پنج الگوریتم Random Forest، Logistic Regression، KNN، SVM و XGBoost بر روی مجموعه‌داده‌ای شامل ۱۰٬۰۰۰ نمونه مورد ارزیابی قرار گرفت. معیار اصلی ارزیابی، Macro-F1 به‌همراه ماتریس درهم‌ریختگی، دقت (Accuracy)، Precision و Recall بوده است.

بر اساس نتایج به‌دست‌آمده، الگوریتم XGBoost با مقدار Macro-F1 حدود 0.8365 بهترین عملکرد را در میان مدل‌های بررسی‌شده نشان داد. الگوریتم KNN عملکردی بسیار نزدیک به XGBoost داشته است (Macro-F1 حدود 0.8350)، در حالی که Logistic Regression با دقت کلی 0.78 و Macro-F1 برابر با 0.76 نسبت به دو مدل قبلی عملکرد ضعیف‌تری ارائه داده است.

تحلیل شاخص‌های Precision و Recall نشان می‌دهد که XGBoost توانسته تعادل مناسبی میان دو کلاس برقرار کند و در کلاس مثبت (1) نیز Recall بالایی داشته باشد. با توجه به اینکه معیار Macro-F1 میانگین عملکرد مدل در هر دو کلاس را بدون توجه به توزیع داده‌ها نشان می‌دهد، این مدل از نظر تعادل و پایداری عملکرد، برتری نسبی خود را حفظ کرده است.

در خصوص الگوریتم SVM باید توجه داشت که به دلیل پیچیدگی محاسباتی و زمان‌بر بودن فرآیند آموزش، این مدل بر روی تعداد کمتری از داده‌ها آموزش داده شده است. این موضوع می‌تواند منجر به کاهش قابلیت تعمیم‌پذیری و در نتیجه افت دقت نهایی شده باشد. در صورت آموزش SVM بر روی کل داده‌ها و با تنظیم مناسب ابرپارامترها، احتمال بهبود عملکرد آن وجود دارد.

در مجموع، با در نظر گرفتن معیارهای ارزیابی، تعادل میان Precision و Recall و همچنین کارایی محاسباتی، مدل XGBoost به‌عنوان گزینه نهایی و بهینه برای این مسئله انتخاب می‌شود؛ چرا که علاوه بر دستیابی به بالاترین مقدار Macro-F1، عملکردی پایدار و متوازن در هر دو کلاس ارائه داده است.