

## مقدمه :

صنعت اتومبیل هر روزه در حال پیشرفت است و برند های مختلف از تمام کشور های دنیا در رقابت با یکدیگر هر ساله در حال معرفی کردن اتومبیل های جدیدشان با امکانات هر چه بیشتر و پیشرفته تری هستند. با توجه به اطلاعات ثبت شده در International Organization of Motor Vehicle Manufacturers تعداد اتومبیل های در حال استفاده در جهان بیش از ۱.۲ بیلیون تا سال ۲۰۱۴ میلادی اعلام شده است. همینطور هر ساله بیش از ۱۰۰ میلیون اتومبیل تولید می شود که تقریباً ۸۴٪ آن برای استفاده های شخصی است. این ارقام بالا اهمیت بسیار بالای مساله خرید و فروش اتومبیل ها را نشان می دهند .



با توجه به گران شدن قیمت خودرو در سال های اخیر تقاضا برای خرید خودرو های کار کرده نیز بیشتر شده و بازار خودرو های کار کرده بسیار گسترده تر از خودرو های صفر است. بدیهی است که قیمت یک فاکتور بسیار مهم در این معاملات بوده است. برخلاف خودروی صفر که قیمت آن توسط شرکت های سازنده تعیین و کنترل می شوند ، معاملات مربوط به خودرو های کار کرده در بازار ها با قیمت های متفاوت صورت می گیرد. با توجه به ویژگی های مختلف تأثیر گذار روی قیمت اتومبیل کار کرده اعم از کیلومتر طی شده ، رنگ ماشین ، زدگی های آن و تعداد آن ها ، اتومات یا دنده ای بودن ، رنگ خوردگی ، نوع سوخت مصرفی ، شرکت سازنده ، سال تولید آن و بسیاری از دیگر ویژگی ها تعیین قیمت اتومبیل کار کرده چندان کار ساده ای نیست و معیار مشخصی در بازار ها خصوصاً در بازار اتومبیل ایران برای آن وجود ندارد.

از این رو قیمت خودرو کارکرده برای عموم مردم همواره دارای ابهام بوده و توافقی تعیین می گردد. از این رو وجود سیستمی که بتواند با معیار های مناسب قیمت این خودرو ها را با توجه به شرایط آنها پیش بینی کند به شدت احساس می شود.

استفاده از روش های تخمین مقدار یک متغیر بر اساس اطلاعات موجود، همواره دغدغه ی اندیشمندان علوم مختلف بوده است. در این شرایط بدیهی است روش هایی دارای ماندگاری بیشتر و کاربردی تر هستند که دارای خطای کمتری باشند. از این رو در سال های بسیار، روش های ریاضی؛ اعم از میانگین ساده، میانگین موزون، میانگین دوبل، رگرسیون و مانند اینها، تنها الگو هایی بودند که مورد استفاده قرار می گرفتند. اما در مواقع گوناگون دارای اشکالاتی نیز بودند که مساله ها رو حل نشده باقی می گذاشتند.

یادگیری ماشین که در ادامه درباره ی تاریخچه ی آن هم توضیحاتی داده می شود مبحثی است که در این شرایط به ما کمک کرد که بتوانیم این فرآیند را هوشمند کرده و انسان را از این پروسه حذف کنیم تا نه تنها در زمان صرفه جویی کنیم بلکه دقت را افزایش دهیم. یادگیری ماشین دارای روش های گوناگونی است که هر یک مزایا و معایب خود را دارند و با توجه به اطلاعات مساله می توان درباره انتخاب آنها با توجه به مساله تصمیم گیری کرد.

در این مطالعه قصد داریم عمل کرد دوروش یادگیری ماشین یعنی Linear Regression و Neural Network را برای تخمین قیمت اتومبیل کارکرده بررسی کنیم. لازم به ذکر است که روش های دیگر یادگیری ماشین مانند Suport Vector Machine و Random Forest هم می توانند عمل کرد نسبتاً خوبی در تخمین قیمت داشته باشند که در اینجا بررسی نشده اند.

اطلاعات استفاده شده در اینجا از سایت دیوار crawl شده و اطلاعات خام شامل ۳۷۱۸۱ مورد ماشین پر استفاده تر در بازار اتومبیل ایران است که پس از تمیز کردن این اطلاعات فرآیند یادگیری روی ۱۱۳۷۲ تعداد ماشین انجام شده است. در ابتدا نگاهی کوتاه به مفاهیم و تاریخچه ی یادگیری ماشین و روش های آن می اندازیم.

## تاریخچه ی یادگیری ماشین :

یادگیری ماشین شاخه ای از علوم کامپیوتر است که بدون انجام برنامه نویسی صریح، به کامپیوتر توانایی یادگیری می بخشد. آرتور ساموئل (Arthur Samuel) امریکایی، یکی از پیشروهای حوزه بازی های کامپیوتری و هوش مصنوعی، عبارت “یادگیری ماشین” را در سال ۱۹۵۹ که در IBM کار می کرد، به ثبت رساند. یادگیری ماشین، که از الگوشناسی و نظریه یادگیری محاسباتی الهام گرفته شده است، مطالعه و ساخت الگوریتم هایی را که می

توانند بر اساس داده ها یادگیری و پیش بینی انجام دهند بررسی می کند - چنین الگوریتم هایی از دستورات برنامه پیروی صرف نمی کنند و از طریق مدلسازی از داده های ورودی نمونه، پیش بینی یا تصمیم گیری می کنند. یادگیری ماشین در کارهای محاسباتی که طراحی و برنامه نویسی الگوریتم های صریح با عملکرد مناسب در آن ها سخت یا نشدنی است، استفاده می شود؛ برخی کاربردها عبارت اند از فیلترینگ ایمیل، شناسایی مزاحم های اینترنتی یا بدافزارهای داخلی که قصد ایجاد رخنه اطلاعاتی دارند، نویسه خوان نوری (OCR)، یادگیری رتبه بندی، و بینایی ماشین.

یادگیری ماشین ارتباط نزدیکی با آمار محاسباتی دارد (و اغلب با آن هم پوشانی دارد)، تمرکز این شاخه نیز پیش بینی کردن توسط رایانه است و پیوند محمکی با بهینه سازی ریاضی دارد، که آن هم روش ها، تئوری ها و کاربردهایی را وارد میدان می کند. یادگیری ماشین گاهی اوقات با داده کاوی ادغام می شود؛ تمرکز این زیرشاخه بر تحلیل اکتشافی داده ها است و با عنوان یادگیری بی نظارت شناخته می شود. یادگیری ماشین نیز می تواند بی نظارت باشد و برای یادگیری و شناخت فرم ابتدایی رفتار موجودات مختلف و سپس پیدا کردن ناهنجاری های معنادار استفاده شود.

در زمینه تحلیل داده ها، یادگیری ماشین روشی برای طراحی الگوریتم ها و مدل های پیچیده است که برای پیش بینی استفاده می شوند؛ در صنعت این مطلب تحت عنوان تحلیل پیشگویانه شناخته می شود. این مدل های تحلیلی به محققان، پژوهشگران علم داده ها، مهندسان و تحلیلگران اجازه می دهد “تصمیمات و نتایجی قابل اطمینان و تکرارپذیر بدست آورند” و با یادگیری از روابط و روندهای مربوط به گذشته، از “فراست های پنهان” پرده برداری کنند.

### تعریف دقیق تر از الگوریتم های یادگیری ماشین :

تام ام. میچل (Tom M. Mitchell) تعریفی پر کاربرد و صوری از الگوریتم های مورد مطالعه در حوزه یادگیری ماشین ارائه نمود: “گوییم یک برنامه کامپیوتری از تجربه E نسبت به یک کلاس T از کارها و اندازه عملکرد P، یاد گرفته است، هرگاه با داشتن تجربه E عملکرد آن که توسط P اندازه گیری می شود در کارهای کلاس T بهبود یافته باشد.” این تعریف از کارهایی که یادگیری ماشین درگیر آن است، تعریفی کاملاً اجرایی است و نه صرفاً تعریفی شناختی. این تعریف دنباله رو پروپزال آلن تورینگ (Alan Turing) در مقاله او “هوش و ماشین محاسبه گر” است

که در آن، سوال “آیا ماشین ها می توانند فکر کنند؟” با سوال “آیا ماشین ها می توانند کاری را انجام دهند که ما (به عنوان موجودات متفکر) می توانیم انجام دهیم؟” جایگزین شد.



### ارتباط یادگیری ماشین و هوش مصنوعی :

یادگیری ماشین از حوزه هوش مصنوعی فراتر است. در همان روزهای ابتدایی ایجاد هوش مصنوعی به عنوان رشته ای علمی، برخی محققان در پی این بودند که ماشین ها از داده ها یادگیری کنند. آن ها تلاش کردند این مسئله را با روش های نمادین متنوعی، و نیز چیزی که آن موقع “شبکه های عصبی” نام داشت، حل کنند؛ این روش ها اغلب پرسپترون (perceptron) و مدل های دیگری بودند که بعد ها مشخص شد بازطراحی مدل های خطی تعمیم یافته آماری بوده اند. استدلال احتمالاتی، به ویژه در تشخیص پزشکی مکانیزه، مورد استفاده قرار گرفت.

با این حال، تاکید روز افزون بر روش منطقی و دانش-محور، شکافی بین AI (هوش مصنوعی) و یادگیری ماشین ایجاد کرد. سیستم های احتمالاتی پُر شده بودند از مسائل تئوری و عملی در مورد بدست آوردن و نمایش داده ها. تا سال ۱۹۸۰، سیستم های خیره بر AI رجحان یافتند و آمار دیگر مورد توجه نبود. کار بر روی یادگیری نمادین/دانش-محور، درون حیطه AI ادامه پیدا کرد و به برنامه نویسی منطقی استقرایی منجر شد، اما سیر آماری

پژوهش دیگر از حیطه AI صرف خارج شده بود و در الگوشناسی و بازیابی اطلاعات دیده می شد. پژوهش در زمینه شبکه های عصبی نیز حدود همین زمان توسط AI و علوم کامپیوتر (CS) طرد شد. این مسیر نیز خارج از حوزه AI/CS توسط محققان رشته های دیگر از جمله هاپفیلد (Hopfield)، راملهارت (Rumelhart) و هینتون (Hinton) تحت عنوان پیوندگرایی (connectionism) دنبال شد. موفقیت عمده آن ها در اواسط دهه ۱۹۸۰ با بازتولید پس نشر (backpropagation) حاصل شد.

یادگیری ماشین، پس از احیا به عنوان رشته ای مجزا، در دهه ۱۹۹۰ شروع به درخشش کرد. این رشته هدف خود را از دستیابی به هوش مصنوعی، به درگیر شدن با مسائل حل پذیری که طبیعتی عملی دارند، تغییر داد و تمرکز خود را از روش های نمادینی که از هوش مصنوعی به ارث برده بود، به روش ها و مدل هایی که از آمار و احتمالات قرض گرفته بود، انتقال داد. این رشته همچنین از اطلاعات دیجیتالی که روز به روز دسترس پذیر تر می شدند و از امکان توزیع آن ها در اینترنت، بهره برد.

## ارتباط یادگیری ماشین و بهینه سازی :

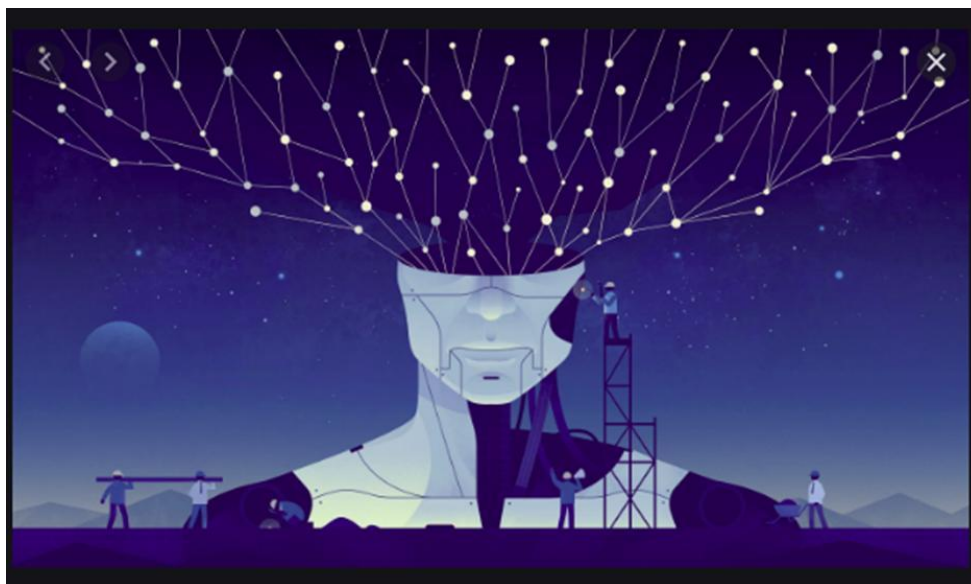
یادگیری ماشین ارتباط تنگاتنگی با بهینه سازی دارد: بسیاری از مسائل یادگیری به شکل مینیمم سازی یک تابع زیان روی یک مجموعه از مثال های آموزشی بیان می شوند. توابع زیان، بیان کننده اختلاف بین پیش بینی های مدل تحت یادگیری و شواهد واقعی مسئله هستند (برای مثال، در طبقه بندی، هدف تخصیص برچسب به شواهد است، و به مدل ها آموزش داده می شود تا قبل از تخصیص، برچسب های یک مجموعه از مثال ها را پیش بینی کنند). تفاوت میان این دو رشته، از هدف کلان آن ها نشئت می گیرد: در حالیکه الگوریتم های بهینه سازی می توانند زیان را روی یک مجموعه آموزشی کمینه کنند، یادگیری ماشین می خواهد زیان را روی نمونه های مشاهده نشده کمینه کند.

## تئوری یادگیری ماشین :

یک هدف اساسی ماشین یادگیرنده، تعمیم دهی از تجربه است. منظور از تعمیم دهی در این چهارچوب، توانایی یک ماشین یادگیرنده در داشتن عملکردی دقیق در فعالیت ها و مثال های جدید و دیده نشده، بر مبنای تجربه آن ماشین با مجموعه داده های آموزش است. مثال های آموزشی از یک توزیع عموماً ناشناخته می آیند (که به عنوان نماینده

فضای رخدادهادر نظر گرفته می شود) و یادگیرنده باید برای این فضا مدلی عمومی تولید کند که به آن، توانایی پیشبینی بقدر کافی دقیق در موارد جدید را بدهد.

حالا می خواهیم به صورت دقیق تر نگاهی به روش شبکه های عصبی مصنوعی بیاندازیم که یکی از شاخه های یادگیری عمیق محسوب می شود :



### تاریخچه و مفاهیم مرتبط با یادگیری عمیق و شبکه های عصبی مصنوعی:

یادگیری عمیق، دسته ای از الگوریتم های یادگیری ماشین است که:

۱- از آشنایی از لایه های چندگانه واحدهای پردازش غیرخطی برای استخراج و تبدیل ویژگی استفاده میکنند. هر لایه تالی، از خروجی لایه قبل به عنوان ورودی استفاده میکند.

۲- به شکلی نظارت شده (مثل طبقه بندی) و یا بدون نظارت (مثل تحلیل الگو) یادگیری میکنند.

۳- لایه های چندگانه ای از نمایش را یادگیری میکنند که متناظر با سطوح مختلفی از انتزاعات هستند؛ این سطوح سلسله ای از مفاهیم را تشکیل میدهند.

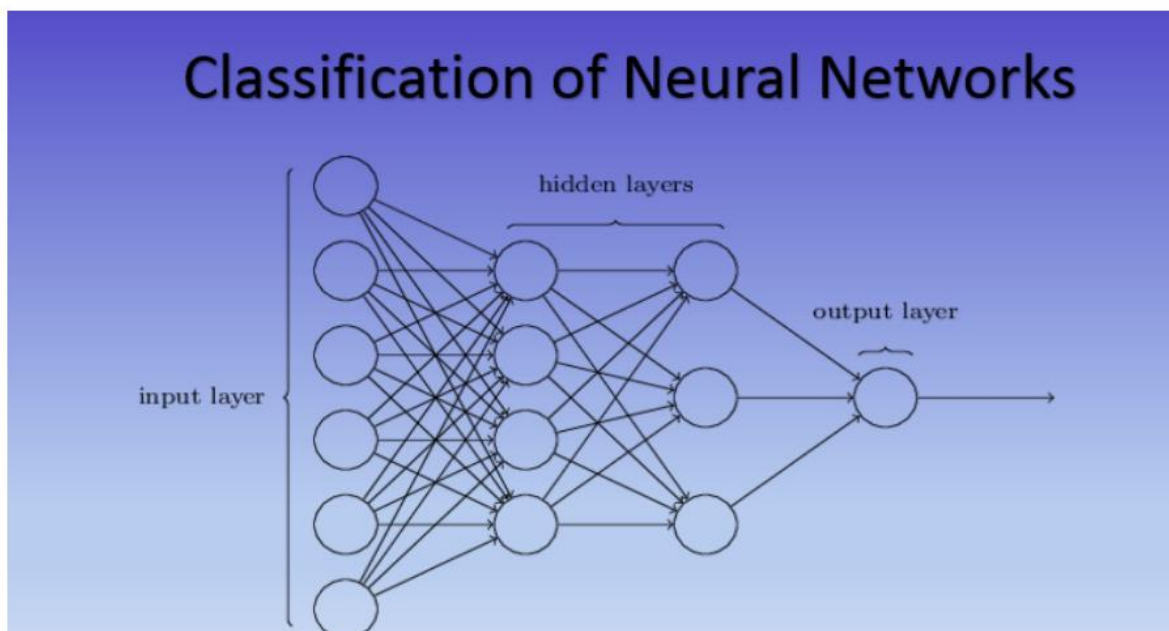
در یادگیری عمیق، هر سطح یاد میگیرد که داده های ورودی خود را به یک نمایش اندکی مجردتر و ترکیبی تر تبدیل کند. در یک کاربرد شناسایی تصویر، ورودی خام میتواند ماتریسی از پیکسل ها باشد؛ اولین لایه نمایشی ممکن است پیکسل ها را مجرد کند و لبه ها را کدگذاری کند؛ لایه دوم ممکن است چینش لبه ها را بسازد و کدگذاری کند؛ لایه سوم ممکن است بینی و چشم ها را کدگذاری کند؛ و لایه چهارم ممکن است تشخیص دهد که تصویر، شامل یک چهره است. چیزی که اهمیت دارد، این است که یک پروسه یادگیری عمیق، به خودی خود میتواند یاد بگیرد که کدام ویژگی ها بطور بهینه در کدام سطح قرار دهد.

### شبکه های عصبی مصنوعی :



شبکه های عصبی مصنوعی (ANN) یا سیستم های اتصالگر، سیستم های محاسبه گری هستند که از شبکه های عصبی زیستی تشکیل دهنده ذهن حیوانات الهام گرفته شده اند. این سیستم ها، با بررسی مثال ها، فعالیت ها را یادگیری می کنند (به عبارت دیگر عملکرد خود را در در انجام فعالیت ها به مرور بهبود می دهند) و عموماً این اتفاق بدون هیچ برنامه نویسی مختص به فعالیتی انجام می شود. برای مثال، در شناسایی تصویر، این شبکه ها می توانند یاد بگیرند که تصاویر شامل گربه را با تحلیل تصاویر مثالی که قبلاً بطور دستی به عنوان “با گربه” یا “بدون گربه” برچسب گذاری شدند، شناسایی کنند و از این نتایج تحلیلی برای شناسایی گربه در تصاویر دیگر استفاده نمایند. این شبکه ها بیشترین استفاده را در کاربردهایی دارند که بیان آنها با یک الگوریتم سنتی که از برنامه نویسی قاعده-بنیان استفاده میکند، دشوار است.

## ساختمان یک شبکه عصبی مصنوعی :



یک **ANN** بر مجموعه ای از واحدهای متصل یا گره، به نام نورون های مصنوعی (مشابه نورون های زیستی در یک مغز زیستی)، مبتنی است. هر اتصال (سیناپس) میان نورون ها می تواند سیگنالی را از یک نورون به نورون دیگر انتقال دهد. نورون دریافت کننده (پست سیناپتیک) می تواند سیگنال (ها) و سپس نورون های پایین دستی سیگنال متصل به آن (ها) را پردازش کند. نورون ها ممکن است دارای حالت باشند، که معمولاً با اعداد حقیقی بین ۰ و ۱ نمایش داده میشود. نورون ها و سیناپس ها همچنین ممکن است وزن داشته باشند که با پیشرفت یادگیری، تنظیم می شود. این وزن، قدرت سیگنالی را که به نورون های پایین دستی فرستاده میشود، افزایش یا کاهش می دهد.

معمولاً نورون ها در لایه ها سازماندهی می شوند. لایه های مختلف ممکن است تبدیلات مختلفی روی ورودی خود، اعمال کنند. سیگنال ها از اولین لایه (ورودی) به آخرین لایه (خروجی) سفر می کنند، و در این بین ممکن است لایه هایی را چند بار طی کنند.

هدف آغازین رویکرد شبکه های عصبی، حل مسئله به روش ذهن انسان بود. با مرور زمان، توجه صرفاً روی برابری با برخی توانایی های خاص ذهنی معطوف شد، و به انحرافات از زیست شناسی، مثل پس-نشر، یا انتقال اطلاعات در جهت عکس و تنظیم شبکه برای انعکاس این اطلاعات، منجر شد.



## کاربرد آنها :

شبکه های عصبی در فعالیت های متنوعی استفاده شده اند، از جمله بینایی رایانه، شناسایی گفتار، ترجمه ماشینی، فیلترینگ شبکه های اجتماعی و بازی های ویدیویی و تشخیص پزشکی.

تا سال ۲۰۱۷، شبکه های عصبی معمولاً از چند هزار تا چند میلیون واحد و چند میلیون اتصال برخوردار هستند. گرچه این عدد به مراتب کوچکتر از تعداد نورون های مغز انسان است، اما این شبکه ها میتوانند فعالیت های زیادی را در سطح فرا انسانی انجام دهند (مثل شناسایی چهره، بازی "Go" و غیره).

## معرفی شبکه های عصبی DNN , RNN , CNN :

شبکه عصبی عمیق (DNN)، یک نوع شبکه عصبی مصنوعی (ANN) با لایه های متعددی بین ورودی و خروجی است. DNN روابط ریاضی صحیح را برای تبدیل ورودی به خروجی پیدا میکند، خواه این روابط خطی باشند خواه غیرخطی. شبکه با حرکت در لایه ها، احتمال هر خروجی را محاسبه میکند. برای مثال، DNNی که آموزش دیده تا نژادهای سگ را تشخیص دهد، تصویر داده شده را بررسی و احتمال اینکه سگ داخل تصویر، نژادی خاص باشد را محاسبه میکند. کاربر میتواند نتایج را بررسی و تعیین کند که شبکه چه احتمالاتی را باید نشان دهد (مثلاً احتمالات بالاتر از یک مقدار خاص و غیر) و برچسب پیشنهادی را بازگرداند. هر محاسبه ریاضی این چینی را به عنوان یک لایه در نظر میگیرند، و DNNهای پیچیده لایه های زیادی دارند، لذا نام شبکه های "عمیق" برایشان انتخاب شده است. هدف نهایی این است که شبکه ای آموزش داده شود تا تصویر را به ویژگی های آن تجزیه، روندهای موجود در تمام نمونه ها را شناسایی، و تصاویر جدید را طبق شباهت هایشان بدون نیاز به ورودی انسانی طبقه بندی کند.

DNNها میتوانند روابط پیچیده غیرخطی را مدل سازی کنند. معماری های DNN، مدل هایی ترکیبی تولید میکنند که در آن شیء به عنوان ترکیبی لایه ای از داده های اولیه بیان میشود. لایه های اضافی، ترکیب ویژگی های لایه های پایین تر را ممکن میسازند، که بطور بالقوه موجب مدل سازی داده ها با واحدهایی کمتر از یک شبکه کم عمق با عملکرد مشابه میشود.

معماری های عمیق شامل اشکال متعددی از چند روش اساسی هستند. هر معماری در زمینه ای خاص موفق بوده است. مقایسه عملکرد چند معماری همواره ممکن نیست، مگر اینکه روی یک مجموعه داده ارزیابی شوند.

**DNN**ها معمولاً شبکه هایی پیشخور هستند که در آن داده ها از لایه ورودی، بدون حلقه، به سمت لایه خروجی جریان پیدا میکند. ابتدا **DNN** نگاهی از نورون های مجازی درست میکند و به اتصالات میان آنها، مقادیر عددی تصادفی یا “وزن” تخصیص میدهد. وزن ها و ورودی ها ضرب میشوند و یک خروجی بین ۰ و ۱ را بازمیگردانند. اگر شبکه بطور دقیق الگوی مورد نظر را تشخیص ندهد، یک الگوریتم وزن ها را تنظیم میکند. به این طریق الگوریتم میتواند تاثیر برخی پارامترها را بیشتر کند، تا وقتی که محاسبات ریاضی صحیح را برای پردازش کامل داده ها پیدا کند.

شبکه های عصبی بازگشتی (**RNN**)، که در داده های آنها میتوانند در هر جهتی جریان پیدا کنند، برای کاربردهایی مثل مدل سازی زبان استفاده میشوند. حافظه کوتاه مدت بلند بطور ویژه ای برای این مصرف اثربخش است.

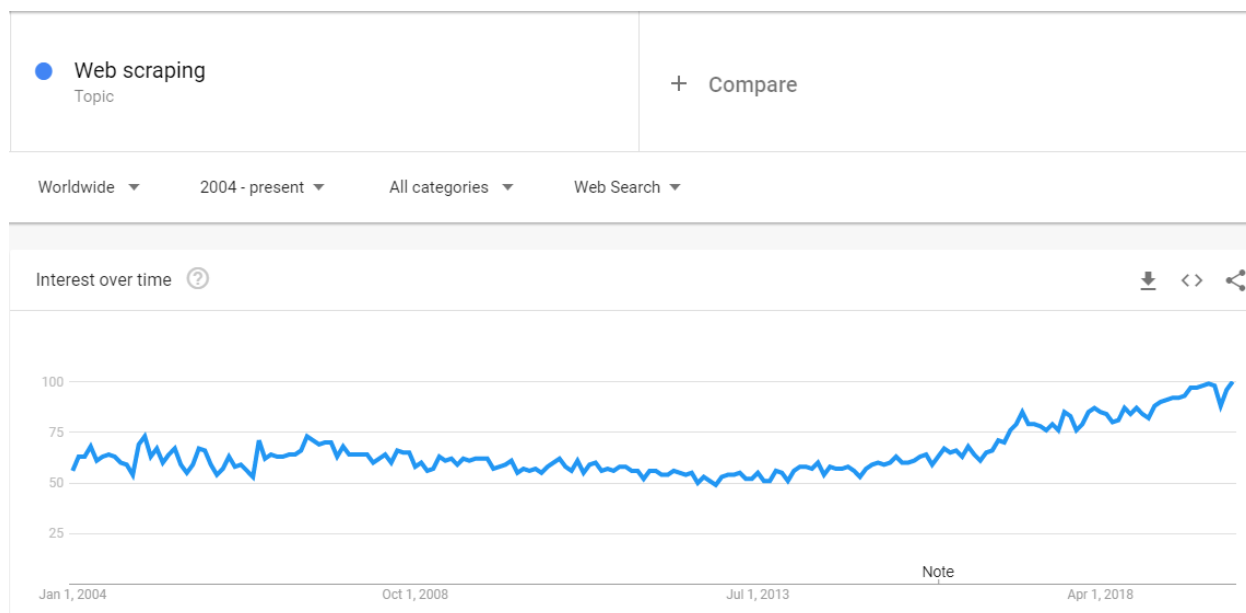
شبکه های عصبی عمیق پیچشی (**CNN**) در بینایی رایانه استفاده میشوند. **CNN**ها همچنین در مدل سازی آکوستیک برای شناسایی گفتار خودکار (**ASR**) استفاده میشوند.

در این پروژه ما قصد داریم از شبکه های عصبی **DNN** استفاده کنیم.

## جمع آوری داده ها :

امروزه با رشد اینترنت و فضای مجازی و وابسته شدن بسیاری از کسب و کار ها به فضای مجازی یکی از مهم ترین عوامل برای موفقیت یا شکست این کسب و کار ها و بسیاری عوامل دیگر منوط به داشتن داده های دقیق و ارزشمند به مقدار کافی میباشد. داده ها امروزه برای شرکت های بزرگ تصمیم گیری میکنند و سیاست های مختلف آن ها را رقم میزنند. به دست آوردن دیتای معتبر که بتوان روی آن تحقیقات مختلفی انجام داد کار بسیار سختی است اما برای موفقیت نیاز است که بتوانیم معتبر ترین و جدید ترین اطلاعات در زمینه های مختلف را به صورت طبقه بندی شده به دست بیاوریم تا بتوانیم از آن ها نتیجه گیری کنیم ، در بسیاری از موارد شرکت ها یا دارندگان این اطلاعات علاقه ای به ، به اشتراک گذاشتن این اطلاعات با دیگران ندارند به همین دلیل امروزه یکی از مهم ترین ابزار هایی که برای هر متخصص علوم داده ضروری است که به آن تسلط داشته باشد، ابزار های مربوط به **Crawl** کردن داده میباشد.

تصویری که در زیر قرار دارد از Google Trends گرفته شده است و نموداری که میبینید نشان دهنده مقدار Search شدن عبارت web scraping در گوگل میباشد که همانطور که مشاهده میکنید در حال افزایش است و این امر نشان دهنده اهمیت بسیار زیاد این ابزار ها در دنیای کسب و کار و تجارت میباشد.



## خزنده وب چیست؟

خزنده وب یا Web Crawler ابزاری است که فرایند دریافت داده های موجود در صفحات وب را خودکار می کند، برای مثال دوست دارید لیست قیمت محصولات دیجیکالا را بصورت روزانه دریافت کنید و تغییرات را مقایسه کنید ، دیجیکالا هیچ API ای برای دسترسی به اطلاعات محصولاتش در اختیار عموم قرار نمیده ، پس یا باید به صورت دستی قیمت ها رو کپی کنیم یا ابزاری داشته باشیم که این کار به صورت خودکار انجام بده ، خزنده های وب دقیقا برای این منظور ساخته شده اند. خزنده وب محتوای وبسایت را دانلود می کند و بخش هایی از سایت که محتوای مورد نظر ما اونجا قرار دارد رو استخراج میکند.

## نحوه کار

به صورت عمومی نحوه کار Web crawler ها به این صورت است که ابتدا لیستی از URL ها (آدرس های وب) که به عنوان seed شناخته می شوند را برای بازدید پردازش می کنند. هنگام پردازش این آدرس ها، لیست لینک

ها و آدرس های موجود در صفحات آن ها را گردآوری کرده و به لیست ابتدایی اضافه می کنند. بقیه اطلاعات را نیز با توجه به نیاز و هدف خود ذخیره و پردازش می نمایند.

در زبان python کتابخانه ها و framework های مختلفی برای خزیدن صفحات وب و استخراج داده های مورد نظر از آن ها وجود دارد که از قدرتمند ترین آن ها میتوان به BeautifulSoup و Scrapy اشاره کرد.



## Scrapy چیست؟

فرض کنید برای انجام یک پروژه مجبور هستید اطلاعات یا داده را از روی صفحه وب استخراج کنید. اگر این داده ها در مقیاس بزرگ باشند کار کمی مشکل ساز می شود. با افزایش حجم داده و توسعه وب، نیاز به تکنیک هایی که بتوانند دستیابی به داده ها و استخراج آنها را فراهم کنند پدیدار می شوند. در این مواقع میتوان از Scrapy استفاده کرد. فریمورک Scrapy یک ابزار قدرتمند برای استخراج داده از وب می باشد. با استفاده از Scrapy می توانید هر نوع اطلاعات را از سایت های مختلف استخراج کرده و در پایگاه داده خود ذخیره فرمایید. برای مثال می توانید کتاب های یک کتابخانه یا عکس های یک گالری و یا اطلاعات ویکی پدیا را استخراج کنید.

این فریمورک سریع با زبان برنامه نویسی پایتون نوشته شده است که به راحتی قابل گسترش، متن باز و رایگان، چندسکویی بوده و دارای قابلیت های گوناگون است. برای استخراج داده از BeautifulSoup هم می توانید استفاده کنید. البته Scrapy امکانات بیشتری را در اختیار شما می گذارد.

برای استفاده از Scrapy همان طور که گفته شد نیاز است که ما Spider ای طراحی کنیم که یک وبسایت به عنوان seed بگیرد و سپس لینک های موجود در این صفحه را به شکل و ترتیبی که ما می خواهیم بازدید کند و سپس اطلاعات مورد نظر ما را از این صفحات استخراج کند.

برای این پروژه، داده هایی که ما نیاز داریم شامل اطلاعات آگهی های مربوط به خرید و فروش خودرو های زیر میباشد.

پراید ۱۱۱ - جک s5 - پژو ۲۰۶ تیپ ۵ - پژو ۲۰۶ تیپ ۲ - پژو پارس

دنا - سمند LX - پژو ۴۰۵ - تیبا سدان - پراید ۱۳۱

مبنای ما برای انتخاب کردن این مدل از ماشین ها زیاد بودن تعداد آگهی های مربوط به این ماشین ها و زیاد تر بودن تقاضا برای خرید و فروش در این مدل ها میباشد، با این کار خروجی به دست آمده از تحلیل هایمان روی داده ها ارزش بیشتری خواهد داشت و جمعیت بزرگتری را هدف قرار میدهد و هم این که با این کار مقدار داده هایی که مقادیر خطا دارند نیز راحت تر مشخص میشود و دلیل دیگر نیز این است که شبکه های عصبی برای Train شدن نیاز به تعداد داده های بیشتری نسبت به مدل های خطی یادگیری ماشین دارند به همین دلیل با تعداد داده های کم در یک مدل از ماشین ها، شبکه عصبی ما به خوبی Train نمیشود و به همین دلیل از قدرت پیشبینی خوبی نیز برخوردار نخواهد بود و خطای زیادی خواهد داشت.

اطلاعات مربوط به آگهی های مربوط به این مدل ها را در شهر های پربازدید سایت دیوار که در پایین نمایش داده شده است بررسی میکنیم.

شهرهای پربازدید

اصفهان	شیراز	کرج	مشهد	تهران
رشت	قم	کرمانشاه	تبریز	اهواز

برای جمع آوری داده های مربوط به آگهی ها ابتدا نیاز است که الگوی URL سایت برای این مدل ها و این شهر ها را پیدا کنیم. با کمی دقت به URL سایت در چند مورد از آگهی ها متوجه میشویم که از الگوی زیر پیروی میکند.

```
'https://divar.ir/s/{}/car/{}'
```

که در آن به ازای {} اول نام شهر و به ازای {} دوم مدل ماشین باید از لیست های زیر انتخاب شود.

```
shahr=['tehran','mashhad','karaj','shiraz','isfahan','tabriz','ahvaz','rasht','qom','kermanshah']
```

```
cars=['pride/111','dena/basic','jac/s5','peugeot/206/5','peugeot/206/2',  
      'peugeot/pars/basic','samand/lx','peugeot/405/glx-petrol','tiba/sedan','pride/131']
```

برای مثال اگر آدرس <http://divar.ir/s/karaj/car/peugeot/pars/basic> را باز کنیم با صفحه زیر مواجه میشویم.

خرید و فروش و قیمت خودرو پژو پارس ساده در کرج





 <p>پژو پارس بی رنگ</p> <p>توافقی</p> <p>تیم ساعت پیش در گوهردشت</p>	 <p>پژو پارس</p> <p>توافقی</p> <p>یک ربع پیش در ماهدشت</p>	 <p>خودرو پارس مدل 83</p> <p>جهت معاوضه</p> <p>دقایقی پیش در هشتگرد</p>
 <p>پژو پارس مدل 85</p> <p>توافقی</p> <p>۷ ساعت پیش در مهرشهر</p>	 <p>پژوپارس ۹۳</p> <p>توافقی</p> <p>تیم ساعت پیش در ملارد</p>	 <p>پرشیا مدل ۹۱ با دو لکه رنگ</p> <p>توافقی</p> <p>تیم ساعت پیش در محمد شهر</p>

حال میتوانیم با ساختن یک پروژه با فریمورک Scrapy و ایجاد یک فایل جدید در آن با عنوان Divar.py در آدرس <http://divar.ir/s/karaj/car/peugeot/pars/basic> Crawl/DivarCrawler/Spiders/Divar.py مشخصات Spider خود را در آن تعریف کنیم. Spider ما در ابتدا به عنوان seed از آدرس زیر استفاده میکند.

```
start_urls = ['https://divar.ir/']
```

سپس با تکمیل کردن URL به شکلی که در بالا توضیح داده شد به صفحات مربوط به این مدل خودرو ها در شهر های پر بازدید رجوع میکند و به ازای هر آگهی که در این صفحات موجود بود یکبار تابع parse\_author را صدا میزند. این بخش از کد زمانی است که ما وارد صفحه یکی از آگهی ها شده ایم که به برای مثال به شکل زیر است.



<https://divar.ir/v/gXXtPck-> [لینک به اشتراک گذاری](#)

دیوار هیچ گونه منفعت و مسئولیتی در قبال معامله شما ندارد.

دسته بندی	سواری
محل	کرج، کمالشهر
نوع آگهی	فروشی
برند	پژو پارس ساده
سال ساخت	۱۳۹۲ - ۲۰۱۳
کارکرد	۹۸
وضعیت بدنه	بدون رنگ
رنگ	سفید
گیربکس	دنده ای
نحوه فروش	نقدی
سند	دو برگی
قیمت	۹۵/۵۰۰/۰۰۰ تومان

در این جا میبینیم که اطلاعات زیادی به ازای هر آگهی موجود است که ما نیاز به استفاده از همه ی آن ها نداریم و از آن ها نمیتوانیم برای تخمین قیمت استفاده کنیم و تاثیر معنایی روی قیمت ندارد. تاثیر معنایی در این جا به این معنی است که مثلا ممکن است ماشین هایی که سند دو برگی دارند قیمت بالاتری ا بقیه انواع ماشین ها داشته باشند اما ما میدانیم که رابطه علت و معلولی بین آن ها برقرار نیست به همین دلیل از این ویژگی چشم پوشی میکنیم. به ازای هر آگهی ویژگی های زیر استخراج میشود.

برند - سال ساخت - رنگ - وضعیت بدنه - گیربکس - کارکرد - محل - قیمت

سپس اطلاعات مربوط به هر کدام از مدل ها را درون یک فایل CSV ذخیره میکنیم. به این شکل توانستیم اطلاعات مورد نظرمان از آگهی های موجود در بازار استخراج کنیم.

**: Concat\_cars\_csv.py**

این فایل python به عنوان ورودی فایل های جمع آوری شده توسط spider ها برای مدل های مختلف ماشین را دریافت میکند و با چسباندن این فایل ها زیر هم یک فایل جدید میسازد که تمام این اطلاعات درون آن ها وجود دارد. این فایل شامل اطلاعات ۳۷۱۸۱ آگهی میباشد.

از آن جایی که دیتای به دست آمده از سایت دیوار به شدت کثیف میباشد نیاز است که ابتدا بخشی از آن را در همین بخش تمیز کنیم ، اطلاعات پیش پردازش های اصلی در پوشه preprocess در فایل main.ipynb قرار دارد که جداگانه بررسی خواهد شد.

برای این بخش از پیش پردازش ابتدا نگاهی به داده هایمان می اندازیم.

In [34]: data

Out[34]:

	body	brand	cit	color	gear	price	usage	year
0	بنون رنگ	سند LX EF7	گازسوز	نقره‌ای	دنده‌ای	۱۸۰,۰۰۰	NaN	۲۰۱۰ - ۱۳۸۹
1	بنون رنگ	سند LX EF7	بنزینی	سفید	دنده‌ای	سنور عقب و جلو ، صندلی برقی	۰	۲۰۱۹ - ۱۳۹۸
2	لو لکه رنگ	سند LX EF7	بنزینی	سفید	دنده‌ای	تومان ۸۰,۰۰۰,۰۰۰	سنور عقب و جلو	۲۰۱۸ - ۱۳۹۷
3	بنون رنگ	سند LX	ساده	سفید	دنده‌ای	تومان ۶۷,۵۰۰,۰۰۰	۱۳۵,۰۰۰	۲۰۱۵ - ۱۳۹۴
4	چند لکه رنگ	سند LX	ساده	سفید	دنده‌ای	تومان ۳۵,۰۰۰,۰۰۰	کوثر نوال	۲۰۰۵ - ۱۳۸۴
...	...	...	...	...	...	...	...	...
37176	۸۹,۰۰۰	پژو 206 تیپ ۵	اصفهان، شیخ صدوق شمالی	تومان ۸۱,۰۰۰,۰۰۰	NaN	NaN	NaN	۲۰۱۵ - ۱۳۹۴
37177	بنون رنگ	پژو 206 تیپ ۵	اصفهان، مبارکه	دنده‌ای	سفید	تومان ۸۱,۸۰۰,۰۰۰	سنور عقب و جلو	۲۰۱۴ - ۱۳۹۳
37178	۰	پژو 206 تیپ ۵	تهران، سپهرودی شمالی	توافقی	NaN	NaN	NaN	۲۰۱۹ - ۱۳۹۸
37179	بنون رنگ	پژو 206 تیپ ۵	شیراز، معالی آباد	دنده‌ای	سفید	۷,۰۰۰	NaN	۲۰۱۹ - ۱۳۹۸
37180	بنون رنگ	پژو 206 تیپ ۵	اصفهان، بزرگمهر	دنده‌ای	سفید	تومان ۹۲,۵۰۰,۰۰۰	۸۰,۰۰۰	۲۰۱۶ - ۱۳۹۵

37181 rows × 8 columns

دیتاست مورد نظر در این مساله ، به صورت خام و قبل از تمیز کردن ، دارای ۸ ستون است که هر ستون نشان دهنده ی یک ویژگی برای هر یک از خودرو های موجود در این دیتاست است.

این ویژگی ها شامل { **body** : وضعیت بدنه ، **brand** : شرکت سازنده و نوع و مدل خودرو ، **city** : شهر و منطقه ای که خودرو در آنجاست ، **color** : رنگ خودرو ، **gear** : نوع دنده ی خودرو ، **price** : قیمت معامله شده در بازار ، **usage** : مقدار کیلومتر طی شده توسط خودرو ، **year** : سال تولید خودرو } می شوند . همچنین دارای ۳۷۱۸۱ سطر است که نشان دهنده تعداد خودرو های موجود در این دیتاست است .

همان طور که مشاهده میکنید مقادیر داده های ما در برخی ستون ها برابر NaN است ، این به این معنی است که در این آگهی ها این مقادیر مشخص نشده بودند. از آن جا که ما به اطلاعات تمام ستون ها برای پیش بینی قیمت نیاز داریم پس رکورد هایی که مقادیر یک یا چند تا از ستون های آن ها برابر NaN است را به کمک دستور زیر حذف میکنیم.

```
In [39]: data = data.dropna()
          data.shape
```

```
Out[39]: (19625, 8)
```

همان طور که مشاهده میکنید حدود ۲۰۰۰۰ داده ناقص را به این شکل از دیتاست حذف کردیم. برای ادامه کار دوباره به داده هایمان نگاهی میاندازیم.

```
In [52]: data
```

```
Out[52]:
```

	body	brand	cit	color	gear	price	usage	year
1	بنون رنگ	بنزینی LX EF7 سمد	کرج، کمالشهر	سفید	دندهای برقی	سنسور عقب و جلو ، صندلی برقی	۰	۲۰۱۹ - ۱۳۹۸
2	دو لکه رنگ	بنزینی LX EF7 سمد	شیراز، گلرون	سفید	دندهای	تومان ۸۰,۰۰۰,۰۰۰	سنسور عقب و جلو	۲۰۱۸ - ۱۳۹۷
3	بنون رنگ	ساده LX سمد	کرج، گهر دشت	سفید	دندهای	تومان ۶۷,۵۰۰,۰۰۰	۱۳۵,۰۰۰	۲۰۱۵ - ۱۳۹۴
4	چند لکه رنگ	ساده LX سمد	مشهد، تربت حیریه	سفید	دندهای	تومان ۳۵,۰۰۰,۰۰۰	کوئل نوال	۲۰۰۵ - ۱۳۸۴
5	بنون رنگ	ساده LX سمد	اصفهان، خانه اصفهان	سفید	دندهای	تومان ۲۶,۵۰۰,۰۰۰	۷,۰۰۰	۲۰۱۳ - ۱۳۹۲
...	...	...	...	...	...	...	...	...
37168	بنون رنگ	پژو 206 تیپ ۵	اصفهان، شاهین شهر	نقدی	دندهای	توافقی	۵,۰۰۰	۲۰۱۶ - ۱۳۹۵
37172	بنون رنگ	پژو 206 تیپ ۵	اصفهان، شاهین شهر	سفید	دندهای	تومان ۸۵,۰۰۰,۰۰۰	سنسور عقب و جلو	۲۰۱۵ - ۱۳۹۴
37173	بنون رنگ	پژو 206 تیپ ۵	اصفهان، ملک شهر	سفید	دندهای	تومان ۷۸,۰۰۰,۰۰۰	۹۵,۰۰۰	۲۰۱۵ - ۱۳۹۴
37177	بنون رنگ	پژو 206 تیپ ۵	اصفهان، مبارکه	سفید	دندهای	تومان ۸۱,۸۰۰,۰۰۰	سنسور عقب و جلو	۲۰۱۴ - ۱۳۹۳
37180	بنون رنگ	پژو 206 تیپ ۵	اصفهان، بزرگهر	سفید	دندهای	تومان ۹۲,۵۰۰,۰۰۰	۸,۰۰۰	۲۰۱۶ - ۱۳۹۵

19625 rows × 8 columns



همان طور که از تصویر پیداست متوجه میشویم که مقادیر وارد شده در برخی ستون ها مقادیر درستی نیست و باید این داده ها را نیز حذف کنیم، برای این کار یک لیست از مقادیر معتبر به ازای هر کدام از ستون ها میسازیم و سپس در صورت وجود مغایرت در مقادیر هر یک از ستون ها به ازای رکورد های موجود ، آن رکورد را از دیتا ست حذف میکنیم.

مقادیر معتبر به ازای هر یک از ستون ها به شکل زیر است

```
In [47]: body=['بدون رنگ','چند لکه رنگ','دو لکه رنگ','یک لکه رنگ']
gear=['اتوماتیک','بنده ای']
color=['آبی','آبیالویی','اطلسی','بادمجانی','برنز','پل','بنفش','پوست پیازی','تیتانیوم','خاکستری','خاکي','دلفینی','ذغالی','زرد','زرشکی','زیتونی','سبز','سبزی','سرمه ای','سفید','سفید صدفی','طاللی','طوسی','عنسی','عذایی','قرمز','قهوه ای','کربن•بلک','و
[ 'کرم','گیلانی','منی','مشکی','موکا','نارنجی','نقرآبی','نقره ای','نوک•مدادی','یشمی']
year =['۱۹۸۷ - ۱۳۶۶','۱۹۸۸ - ۱۳۶۷','۱۹۸۹ - ۱۳۶۸','۱۹۹۰ - ۱۳۶۹','۱۹۹۱ - ۱۳۷۰','۱۹۹۲ - ۱۳۷۱','۱۹۹۳ - ۱۳۷۲','۱۹۹۴ - ۱۳۷۳','۱۹۹۵ - ۱۳۷۴','۱۹۹۶ - ۱۳۷۵','۱۹۹۷ - ۱۳۷۶','۱۹۹۸ - ۱۳۷۷','۱۹۹۹ - ۱۳۷۸','۲۰۰۰ - ۱۳۷۹','۲۰۰۱ - ۱۳۸۰','۲۰۰۲ - ۱۳۸۱','۲۰۰۳ - ۱۳۸۲','۲۰۰۴ - ۱۳۸۳','۲۰۰۵ - ۱۳۸۴','۲۰۰۶ - ۱۳۸۵','۲۰۰۷ - ۱۳۸۶','۲۰۰۸ - ۱۳۸۷','۲۰۰۹ - ۱۳۸۸','۲۰۱۰ - ۱۳۸۹','۲۰۱۱ - ۱۳۹۰','۲۰۱۲ - ۱۳۹۱','۲۰۱۳ - ۱۳۹۲','۲۰۱۴ - ۱۳۹۳','۲۰۱۵ - ۱۳۹۴','۲۰۱۶ - ۱۳۹۵','۲۰۱۷ - ۱۳۹۶','۲۰۱۸ - ۱۳۹۷','۲۰۱۹ - ۱۳۹۸']
```

بعد از حذف کردن مقادیر نامعتبر این ستون ها ، داده هایی که مقادیر آن ها در ستون قیمت و کارکرد نیز عددی وارد نشده و دارای حروف است را نیز حذف میکنیم. برای ستون سال نیز ، سال میلادی را از گزینه های حذف میکنیم و فقط سال شمسی را باقی میگذاریم و تمام ارقام فارسی را نیز به انگلیسی تبدیل میکنیم که بتوان از آن ها راحت تر استفاده کرد چون در برخی موارد اعداد فارسی مشکلاتی ایجاد میکنند. پس از انجام تمام این اعمال یکبار دیگر برای اطمینان داده هایی را که مقادیر NaN در آن ها وجود داشته را حذف میکنیم.

توجه داشته باشید که این پیش پردازش ابتدایی و فقط برای حذف کردن داده های زائد بود، برای گرفتن نتیجه خوب روی مدل های یادگیری ماشین و شبکه عصبی نیاز است که وقت و دقت بیشتری را برای انتخاب داده های معتبر و صحیح اختصاص دهیم تا بتوانیم نتیجه دلخواه را از آن ها به دست آوریم.

## پیش پردازش:

بخشی از پیش پردازش داده ها همانطور که مشاهده کردید پس از Crawl کردن داده ها انجام شد و داده های ناقص را از دیتاستمان حذف کردیم حالا وقت آن است که به بررسی دقیق تر داده هایمان بپردازیم و از جهات مختلف آن ها را پالایش کنیم.

نگاهی دیگر به بخشی از داده ها می اندازیم.

year	usage	price	gear	color	cit	brand	body
1394	135000.0	67500000.0	دنده‌ای	سفید	کرج، گوهردشت	ساده LX سمند	بدون رنگ 0
1392	70000.0	26500000.0	دنده‌ای	سفید	اصفهان، خانه اصفهان	ساده LX سمند	بدون رنگ 1
1391	165000.0	57800000.0	دنده‌ای	سفید	شیراز، بلوار رحمت	بنزینی LX EF7 سمند	بدون رنگ 2
1398	13000.0	99500000.0	دنده‌ای	سفید	شیراز، امیرکبیر	بنزینی LX EF7 سمند	بدون رنگ 3
1394	110500.0	63500000.0	دنده‌ای	سفید	شیراز، پل غدیر	ساده LX سمند	دو لکه رنگ 4

- در مرحله ی اول ستون **city** که شامل شهر و منطقه ای که خودرو در آنجا معامله شده است را در نظر گرفته و با توجه به اینکه منطقه چندان تاثیر خاصی روی قیمت خودرو نخواهد داشت و فاکتور مهمی محسوب نمی شود آن را از این ستون حذف کرده ایم. پس در حال حاضر این ستون از دیتاست مورد نظر حذف شده و ستون دیگری به نام **loc\_1** به آن اضافه شده است که فقط شامل نام شهری است که خودرو در آنجا بوده است .

loc_1	year	usage	price	gear	color	brand	body
کرج	1394	135000.0	67500000.0	دنده‌ای	سفید	ساده LX سمند	بدون رنگ 0
اصفهان	1392	70000.0	26500000.0	دنده‌ای	سفید	ساده LX سمند	بدون رنگ 1
شیراز	1391	165000.0	57800000.0	دنده‌ای	سفید	بنزینی LX EF7 سمند	بدون رنگ 2
شیراز	1398	13000.0	99500000.0	دنده‌ای	سفید	بنزینی LX EF7 سمند	بدون رنگ 3
شیراز	1394	110500.0	63500000.0	دنده‌ای	سفید	ساده LX سمند	دو لکه رنگ 4

یکی دیگر از کارهای انجام شده برا تمیز کردن دیتاست این بوده است که اگر دو **syntax** متفاوت در دو قسمت مختلف از دیتاست وجود داشته است که هر دو دارای یک مفهوم مشابه در واقعیت بوده اند ،آنها را یکی کند تا مدل آنها را دو مفهوم متفاوت در نظر نگیرد . مثلا در اینجا تابع **make same** اینکار را برای دو عبارت

'Peugeot 405 GLX-petrol' و 'پژو ۴۰۵ GLX بنزینی' انجام می دهد و هر دوی این عبارات را معادل

'پژو ۴۰۵ GLX بنزینی' در نظر می گیرد.

برای خوانا بودن دیتا ستمان متغیر وابسته ای که قرار است به عنوان **Target** آن را پیش بینی کنیم را جابجا میکنیم و به ستون آخر منتقل میکنیم.

در مرحله بعدی تعداد تکرار هر یک از موارد در ستون های مختلف مانند **year, color, brand** و یا سایر ستون ها را بدست آورده و هر موردی که کمتر از ۱۰ بار تکرار شده باشد را از دیتاست حذف می کنیم:

#### –ستون brand :

هر یک از مدل های خودروهای موجود در دیتاست را بدست آورده و مواردی را که کمتر از ۱۰ بار تکرار شده اند را از دیتاست حذف می کنیم . در اینجا ۵ مورد با این شرایط وجود داشته اند که حذف شده اند . مثلا پیکان بنزینی فقط یک مورد در کل این دیتاست وجود داشته است. علت اینکه بهتر است این موارد حذف شوند این است که فرآیند یادگیری همانطور که قبلا گفته شد با توجه به تجربه و تکرار صورت می گیرد و مثلا برای اینکه مدل ما بتواند قیمت پیکان بنزینی را با دقت بالا پیش بینی کند لازم است تا تعداد بیشتری پیکان بنزینی را تجربه کرده باشد و وجود یک مثال قطعا برای یادگیری آن کافی نبوده و فقط باعث خطای بیشتر مدل روی سایر پیش بینی ها خواهد شد. موارد حذف شده در این ستون:

'پیکان بنزینی', 'تیبا صندوق\دار LX دوگانه\200c سوز', 'تیبا صندوق\دار SX دوگانه\200c سوز', 'تیبا صندوق\دار EX دوگانه سوز', 'تیبا صندوق\دار'

#### –ستون color :

در این ستون نیز پس از بدست آوردن تعداد دفعات تکرار رنگ ها ، ۸ مورد دارای کمتر از ۱۰ تکرار بوده اند که حذف شده اند. موارد حذف شده در این ستون:

['کرم', 'کربن\200c بلک', 'طلایی', 'برنز', 'تیتانیوم', 'نارنجی', 'پوست\200c پیازی', 'زرشکی']

#### –ستون year :

در این ستون نیز تعداد سال های کمتر از ۱۰ بار تکرار شده ۴ مورد بوده است. موارد حذف شده در این ستون:  
[۱۳۶۹, ۱۳۷۵, ۱۳۷۳, ۱۳۷۴]

– ستون های دیگر موارد کمتر از ۱۰ تکرار نداشته اند و بنابراین موردی از آنها حذف نشده است .

۵- تابع `remove_invalid_price` روی ستون `price` فراخوانی شده است تا قیمت هایی که به نامعتبر هستند و واقعی نیستند را تشخیص دهد و آنها را در فرآیند یادگیری وارد نکند. این تابع دو شرط را در ستون قیمت چک می کند. اولاً اینکه اگر رقم پایانی قیمت '۰۰۰' نبود، آن را نامعتبر می داند و همچنین اگر تعداد ارقام این ستون کمتر از ۷ رقم باشد نیز آن را نامعتبر می داند. چرا که این نشان دهنده ی این است که قیمت ماشین کمتر از ده میلیون بوده که این رقم برای قیمت خودرو در سال های اخیر غیر قابل قبول و در نیجه فاقد اعتبار است. تعدادی از قیمت های نامعتبر را در عکس زیر مشاهده می کنیم. در نهایت این تابع ۱۳۰ مورد قیمت نامعتبر را طی این فرآیند شناسایی کرده است.

```
11111111.0
360.0
11111111.0
360.0
360.0
1234567.0
360.0
1111111.0
360.0
360.0
```

نمونه ای از قیمت های نامعتبر شناسایی شده

تا به این مرحله دیتاست مورد نظر از ۱۱۸۷۰ مورد، به ۱۱۶۶۵ مورد تقلیل یافته است. حال کمی با داده های پرت نیز درگیر می شویم و سعی می کنیم تا به داده هایمان را به توزیع نرمال نزدیک تر کنیم تا محاسبات ساده تر و فرآیند یادگیری با دقت بیشتر صورت گیرد

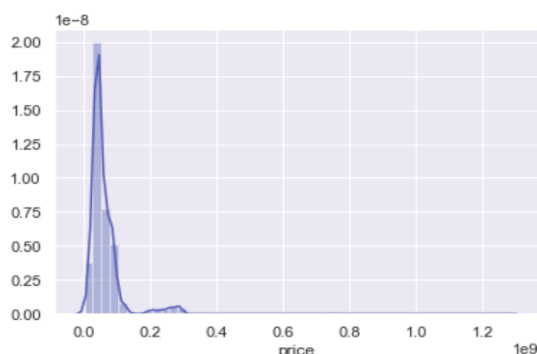
در این قسمت روی داده های پرت در ستون `price` و `usage` کار می کنیم:

برای کار با داده های پرت معمولاً از تابع `quantile` استفاده می کنیم که معمولاً روی داده های عددی فراخوانی می شود و به این صورت است که یک عدد مانند `X` بین صفر تا یک را به عنوان ورودی دریافت کرده و در خروجی عددی را نمایش می دهد که `X` درصد از داده ها مقدار کمتر از خروجی این تابع را داشته اند و درصد باقی مانده از خروجی این تابع مقدار بیشتری داشته اند. در مثال های زیر بهتر متوجه نحوه ی عملکرد این تابع می شویم:

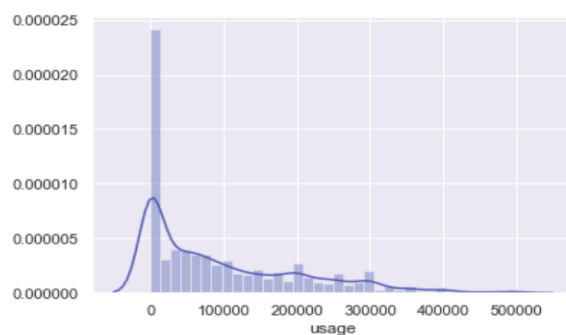
با توجه به توصیفات تابع `describe` برای ستون های `price` و `usage` و همچنین `plot` کردن آنها (که در زیر نشان داده شده اند)، متوجه می شویم که داده های بسیار پرت نیز در این دیتا ها وجود دارند که این دیتا های پرت در

نهایت مدل ما را با خطاهای بسیار بزرگ مواجه می سازند و حداقل امکان باید از وجود آنها در فرآیند یادگیری جلوگیری کرد .

```
count    1.166500e+04
mean     5.991166e+07
std      4.873199e+07
min      1.000000e+06
25%      3.500000e+07
50%      4.680000e+07
75%      7.100000e+07
max      1.280000e+09
Name: price, dtype: float64
```



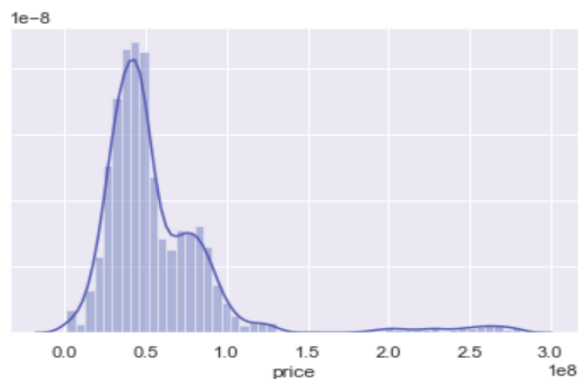
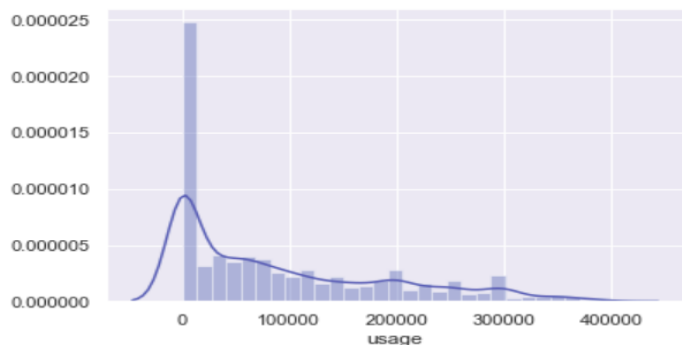
```
count    11548.000000
mean     96289.417388
std      107714.279109
min       0.000000
25%       300.000000
50%      60000.000000
75%     160000.000000
max     500000.000000
Name: usage, dtype: float64
```



برای این کار تابع `quantile(0.99)` را روی این دو ستون فراخوانی می کنیم ، نتایج به شکل زیر است.

```
q = crawled_data['price'].quantile(0.99)
q
282180000.0000003
```

این تصویر نشان میدهد که در بین داده های مربوط به قیمت ۹۹ درصد داده ها مقداری کمتر از ۲۸۲ میلیون تومان دارند و ۱ درصد نهایی داده تا ۱ میلیارد و ۲۰۰ میلیون نیز بالا میروند ، ما برای نزدیک شدن توزیع داده ها به توزیع نرمال از این ۱ درصد چشم پوشی میکنیم و این کار را برای داده های مربوط به کارکرد خودرو نیز انجام میدهیم.



همانطور که در شکل ها نیز مشخص است با حذف داده های پرت ، توزیع داده هایمان به توزیع نرمال نزدیک تر می شوند که این مساله در روش های یادگیری ، بسیار کارآمد است .

تا به این مرحله ، دیتای مورد نظر را تمیز کرده و داده های بسیار پرت را نیز مدیریت کرده و از دیتاست خود حذف کرده ایم . در طول این مراحل تعداد نمونه خودرو ها از ۱۱۸۷۰ به ۱۱۳۷۲ رسیده است یعنی ۴۹۸ داده را به دلایلی که در بالا به تفصیل توضیح داده شد ، حذف کرده و از فرآیند یادگیری خارج کرده ایم .

در نهایت دستور `reset_index` را برای `data` های باقی مانده فراخوانی می کنیم تا به این ۱۱۳۷۲ نمونه دوباره `index` مناسب داده شود . این کار را برای این انجام می دهیم که در طول فرآیند تمیز کردن و حذف کردن داده های نامعتبر یا پرت ممکن است `index` های از قبل داده شده به آنها ترتیب خود را از دست داده باشند و باعث مشکلات احتمالی در آینده شوند .

و در آخر دیتای تمیز شده به صورت زیر خواهد بود:

	brand	color	gear	year	body	loc_1	usage	price
0	LX سمند	سفید	دنده ای	1394	بدون رنگ	کرج	135000.0	67500000.0
1	LX سمند	سفید	دنده ای	1392	بدون رنگ	اصفهان	70000.0	26500000.0
2	LX EF7 سمند	سفید	دنده ای	1391	بدون رنگ	شیراز	165000.0	57800000.0
3	LX EF7 سمند	سفید	دنده ای	1398	بدون رنگ	شیراز	13000.0	99500000.0
4	LX سمند	سفید	دنده ای	1394	دو لکه رنگ	شیراز	110500.0	63500000.0

که به صورت یک جدول ( ۸ , ۱۱۳۷۲ ) تایی است .

دستور آخر در این بخش که بسیار مهم است ، `shuffle` کردن دیتاست است . زیرا در بخش جمع آوری داده ها ، داده ها را به ترتیب مدل ماشین پشت سر هم قرار دادیم و این امر ارزیابی مدل ما را با خطا مواجه میکند یعنی زمانی که قرار است قسمت بندی شوند تا بخشی برای `trian` استفاده شود و بخش دیگر برای `test` ، اگر ترتیب آنها به هم نخورده باشد و دیتا های شبیه هم کنار یکدیگر باشند ، دیتا روی یکسری دیتای نسبتا مشابه `train` می شود و روی بخش دیگری که ویژگی های کاملا متفاوت دارد و روی آن یا مشابه آن ، `trian` نشده ، `test` می شود .

برای این کار دستور زیر را اجرا می کنیم :

```
crawled_data = crawled_data.sample(frac=1).reset_index(drop = True)
```

حال وقت آن است که دیتای تمیز شده را برای یادگیری به ۲ روش regression و neural\_network آماده کنیم . آماده کردن بدین معناست که آنها را متناسب با روش یادگیری encode کنیم چرا که بسیاری از داده های این دیتاست به صورت عددی نیستند و باید برای انجام محاسبات روی آنها ، به داده های کیفی آن نیز کدهایی به صورت عدد نسبت داد که در بخش بعدی بیشتر آن را توضیح می دهیم .

## : Encoding Data






بیشتر الگوریتم های یادگیری ماشین و شبکه های عصبی توانایی کار با داده های کیفی را ندارند و نیاز است که ما این داده های کیفی را به داده های کمی تبدیل کنیم تا مدل های ما بتوانند اعمال ریاضی را روی این اعداد پیاده کنند تا ضرایب و اطلاعات مورد نظر ما را محاسبه کنند. برای Encode کردن داده ها روش های زیادی وجود دارد از جمله Hash Encoding, Count Encoding, One Hot Vector Encoding, Label Encoding و نکته مهمی که راجع به encoding وجود دارد این است که encoding های مختلف روی مدل های مختلف یادگیری ماشین و شبکه عصبی عملکرد متفاوتی دارند. این امر ما را ملزم میکند که برای هر مدلی که میخواهیم استفاده کنیم از Encoding ی بهره بگیریم که تجربیات و مطالعات نشان داده بهترین عملکرد را روی این مدل ها خواهند داشت.

## : Encoding data for Regression

برای Encode کردن داده ها برای مدل رگرسیون ما از One Hot Vector Encoding استفاده میکنیم که از ویژگی های اصلی این نوع encoding این است که در کار با مدل های خطی مانند رگرسیون قدرت بسیار بالایی دارد.

### One hot vector Encoding چگونه داده ها را encode میکند ؟






فرض کنید دیتاستی در اختیار داریم که راجع به ویژگی های افراد مختلف و نسبت آن با حقوق آن ها میباشد. و یکی از ستون های این دیتا ست مربوط به جنسیت این افراد میباشد که در آن مقدار "male" یا "female" به ازای هر رکورد ذخیره شده است. Encode کردن این ستون با One hot Vector به شکل زیر عمل میکند.

Gender		Is_Male	Is_Female
	→	0	1
	→	0	1
	→	1	0
	→	0	1
	→	1	0

یعنی به جای ستون Gender در داده اصلی، به ازای تمام حالت های مختلف موجود در این ستون یک ستون جدید ایجاد میشود ، در این جا چون ستون Gender تنها دو مقدار male و female را میپذیرد پس دو ستون ایجاد میکنیم. یکی is\_male و دیگری is\_female که مقادیر آن ها به شکلی که در تصویر بالا مشاهده میکنید قابل نمایش دادن است.

پس یک ستون که دارای n مقدار کیفی متفاوت باشد را میتوان به کمک n ستون دیگر نمایش داد و تمام داده های کیفی را به داده کمی تبدیل کرد.

یکی از کارهایی که معمولا برای مناسب تر شدن این نوع encoding برای استفاده در مدل های خطی میشود این است که ستون اول این n ستون تولید شده را حذف میکنند. برای مثال بالا خروجی به این شکل میشود.

Gender		Is_Female
	→	1
	→	1
	→	0
	→	1
	→	0



این کار بدون کم کردن توان encoding و از دست دادن اطلاعات، سایز ورودی ما را کاهش میدهد و ستونی را که مقادیر آن کاملاً وابسته به ستون های دیگر بود را حذف میکند که این امر در افزایش کیفیت مدل رگرسیون تاثیر بسزایی دارد.

پس به این شکل میتوان یک ستون که دارای n مقدار کیفی متفاوت است را به کمک n-1 ستون دیگر نمایش داد و داده های کیفی را به کمی تبدیل کرد.

برای حساب کردن تعداد ستون ها در داده هایی که در حال کار کردن روی آن هستیم ابتدا باید تعداد مقادیر منحصر بفرد در هر یک از ستون های کیفی را پیدا کنیم.

تعداد مقادیر منحصر بفرد هر کدام از این ستون ها به شکل زیر است.

```
brand = 28
color = 27
gear = 2
body = 4
location = 10
```

ستون های usage و year و price مقادیر کمی دارند که در این encoding تغییری نخواهند داشت.

تعداد ستون های داده های ما پس از encode شدن به شکل زیر محاسبه میشود.

$$28 - 1 + 27 - 1 + 2 - 1 + 4 - 1 + 10 - 1 + 3 = 69$$

همان طور که در شکل زیر مشاهده میکنید تعداد ستون های داده های encode شده ما برای رگرسیون برابر ۶۹ است.

```
In [27]: reg_encoded.shape
```

```
Out[27]: (11372, 69)
```

برای انجام این نوع encoding روی داده هایمان میتوانیم از دستور زیر استفاده کنیم که در آن drop\_first=True وظیفه حذف کردن همان یک ستون اضافه به ازای هر کدام از ستون های کیفی داده اولیه مان را به عهده دارد.

```
In [10]: reg_encoded = pd.get_dummies(cleaned_data, drop_first=True)
```

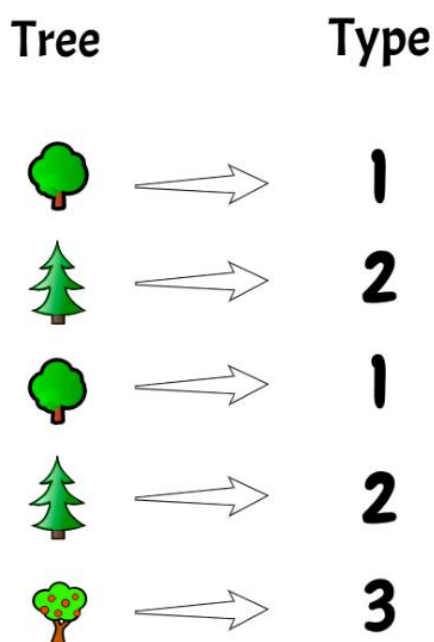
بخشی از داده ها پس از encode شدن به شکل زیر است.

	year	usage	price	brand_تیپا صندوق دار LX	brand_تیپا صندوق دار SX	brand_جک اتوماتیک S5	brand_جک دنده ای S5	brand_دنا معمولی	brand_دنا معمولی 1700cc	brand_سمند LX	...	body_یک تکه رنگ	loc_1_تهران	loc_1_تهران	loc_1_تهران
0	1397	55000.0	70000000.0	0	0	0	0	0	0	0	...	0	0	0	0
1	1392	40000.0	34000000.0	0	0	0	0	0	0	0	...	1	0	0	0
2	1384	300000.0	36500000.0	0	0	0	0	0	0	0	...	0	0	0	0
3	1386	250.0	27200000.0	0	0	0	0	0	0	0	...	1	0	1	0
4	1387	277.0	30000000.0	0	0	0	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

پس از انجام این تغییرات داده های encode شده را با فرمت csv در فایلی با نام reg\_encoded.csv ذخیره میکنیم.

## : Encoding Data for Neural Networks

از one hot vector encoding برای encode کردن داده های برای شبکه عصبی نیز میتوان استفاده کرد اما در این پروژه به دلیل زیاد بودن تعداد مقادیر منحصر بفرد به ازای هر ستون کیفی و عمیق بودن شبکه عصبی که برای این مسئله انتخاب کردیم این نوع encoding عملکرد خوبی نشان نداد به همین دلیل نوع دیگری از encoding را در اینجا مورد استفاده قرار میدهم که Label Encoding نام دارد. نمونه ای از انجام این نوع encoding را در تصویر زیر ملاحظه میکنید.



همان طور که در تصویر واضح است این نوع از encoding به ازای هر عضو منحصر بفرد از یک ستون داده یک عدد در نظر میگیرد و به این ترتیب هر ستون داده کیفی به یک ستون داده کمی تبدیل میشود. این نوع encoding در شبکه های عصبی عمیق عملکرد بسیار خوبی نشان میدهد.

برای مثال برای encode کردن ستون Brand از آن جا که ۲۸ برند منحصر بفرد در داده هایمان داریم پس مقادیر این ستون به ازای تمام رکورد ها برابر مقداری بین ۱-۲۸ خواهد بود.

از آن جایی که به ازای هر ستون داده کیفی یک ستون داده کمی ایجاد میشود پس انتظار داریم تعداد ستون های داده اصلی ما تغییر نکند.

```
In [9]: neu_encoded.shape
```

```
Out[9]: (11372, 8)
```

همان طور که مشاهده میکنید همه چیز مطابق انتظارمان بود.

پس از encode کردن تمام ستون ها به این شکل بخشی از داده نهایی به شکل زیر میباشد.

	brand	color	gear	year	body	loc	usage	price
0	26	6	1	21	0	6	55000.0	70000000.0
1	12	0	1	16	3	5	40000.0	34000000.0
2	24	23	1	8	2	5	300000.0	36500000.0
3	19	15	1	10	3	2	250.0	27200000.0
4	19	10	1	11	2	7	277.0	30000000.0

برای encode کردن این داده ها ابتدا نیاز است که از LabelEncoder را از کتابخانه sklearn، import کنیم.

```
In [4]: from sklearn.preprocessing import LabelEncoder
```

سپس میتوانیم با ساختن یک instance از کلاس LabelEncoder به ازای هر ستون و سپس fit\_transform کردن آن ها روی داده های مورد نظرمون این Encoding را به ازای ستون های مختلف انجام دهیم.

```
In [5]: le = LabelEncoder()
```

```
In [6]: brand = le.fit_transform(cleaned_data['brand'])
```

سپس اطلاعات به دست آمده را با فرمت csv در فایل neu\_encoded.csv ذخیره میکنیم.

حال به مرحله ای رسیده ایم که دیتای مورد نظر خود را تمیز کرده ، داده های پرت را پیدا کرده و حذف کرده ایم . سپس دیتای مورد نظر را encode کرده ایم و دیتا کاملاً آماده است تا به مدل های رگرسیون و شبکه عصبی داده شود .

معرفی کتابخانه های پایتون مورد استفاده در این بخش :



## NumPy

### Numpy

یک کتابخانه برای زبان برنامه نویسی پایتون است. با استفاده از این کتابخانه امکان استفاده از آرایه ها و ماتریسهای بزرگ چند بعدی فراهم میشود. همچنین میتوان از تابع های ریاضیاتی سطح بالا بر روی این آرایه ها استفاده کرد. هدف اصلی NumPy فراهم ساختن امکان کار با آرایه های چندبعدی همگن است. این آرایه ها جدولی از عناصر هستند که همگی از یک نوع میباشند و با یک چندتایی، از اعداد صحیح مثبت اندیسگذاری میشوند. همچنین آرایه های Numpy محاسباتی سریعتر از لیستها دارند و برای اجرای عملیات ریاضیاتی و منطقی بسیار کارآمدتر هستند. به طور کلی این کتابخانه در کار با data بسیار کارآمد بوده و امکانات خوبی را در اختیار ما قرار می دهد تا راحت تر و سریع تر بتوانیم پیش پردازش ها و پردازش ها را انجام دهیم تا به هدف مطلوب برسیم.



### Pandas

یک کتابخانه متن باز پایتون است. این کتابخانه می تواند داده ها را با بهره گیری از ساختارهای Series و DataFrame که ارائه می کند، به قالبی که برای تحلیل داده ها مناسب هستند، مبدل سازد. بسته پانداس حاوی چندین متد برای پالایش مناسب داده ها است. پانداس دارای ابزارهای گوناگونی برای انجام عملیات ورودی خروجی است و می تواند داده ها را از فرمت های گوناگونی شامل MS Excel ، TSV ، CSV و دیگر موارد بخواند.



## Matplotlib

اولین کتابخانه مصورسازی در پایتون که با وجود قدیمی بودن، همچنان یکی از کاربردی ترین ها میباشد کتابخانه matplotlib بسیاری از کتابخانه های دیگری که بعد از این کتابخانه طراحی و ساخته شده اند، دستورهای این کتابخانه را پوشش داده و اجازه میدهند تا برخی از روش های کتابخانه matplotlib را با دستورهای کوتاه تر اجرا کنید، اما با این وجود ضعف هایی نیز دارد که کتابخانه های دیگر، سعی در برطرف کردن آن ها داشته اند.



## Seaborn

این کتابخانه مصورسازی علاوه بر امکان استفاده از دستورات کتابخانه matplotlib در چند خط کد ساده ، دارای سبک ها و پالت های رنگی پیش فرضی است که به شما کمک میکند تا نمودارهای زیباتر و مدرن تری را طراحی کنید.



## Keras

کراس یک فرانت اند (front-end) برای فریمورک های یادگیری عمیق تنسرفلو، CNTK و تیانواست و آن ها پشت شبکه های عصبی را می سازند و آموزش می دهند و برای همین به آن یک چهارچوب سطح بالا می گوییم چون کراس پیچیدگی استفاده از این کتابخانه ها را تا حد خوبی حذف می کند. یک ویژگی خاص دیگر کراس این است که محدود به یک کتابخانه یادگیری عمیق نیست و همانطور که گفتیم می توانیم از تنسرفلو، CNTK و یا تیانو برای محاسبات پشت پرده آن استفاده کنیم.



کتابخانه `scikit-learn` یک کتابخانه پایتون برای یادگیری ماشین است. این کتابخانه برای کاربردهای یادگیری ماشین از جمله خوشه بندی، رگرسیون و کلاس بندی ابزار قدرتمند و در عین حال ساده ای می باشد. یادگیری استفاده از آن بسیار سریع و آسان بوده و انعطاف آن باعث می شود تا بتوان آن را در مسائل متنوعی مورد استفاده قرار داد. روش های متنوعی در آن فراهم آمده است

## روش اعتبارسنجی K-Fold:

در این نوع اعتبارسنجی داده ها به  $K$  زیرمجموعه افراز می شوند. از این  $K$  زیرمجموعه، هر بار یکی برای اعتبارسنجی (Test) و  $K-1$  تای دیگر برای آموزش (Train) بکار میروند. این روال  $K$  بار تکرار می شود و اینگونه مطمئن می شویم که همه داده ها حداقل یکبار برای آموزش و یکبار برای اعتبارسنجی بکار می روند. در نهایت میانگین نتیجه این  $K$  بار اعتبارسنجی به عنوان یک تخمین نهایی برگزیده می شود.

ما در اینجا با توجه به شرایط مساله از `fold-5` استفاده میکنیم.

دیتای خود را بر اساس `index` هایشان، در ابتدا به ۵ دسته ی مساوی تقسیم می کنیم. این بخش ها را با نام های 'five', 'four', 'three', 'two', 'one'، مشخص می کنیم و داده های `test` و `train` را در هر بخش تعیین می کنیم. پس از انجام این کار، سائز داده های `train` در هر `fold`، تقریباً برابر ۸۹۰۹ و سائز داده های `test` برابر ۴۷۲۲ نمونه بدست آمده است.

با توجه به اینکه این دسته بندی بر اساس `index` ها بوده است، الزم است تابعی تعریف کنیم تا `index` ها را به مقادیر موجود در دیتاست به صورت `X` و `Y` تبدیل کند که `X` هر نمونه، لیستی است که در واقع همان `Y` ویژگی های موجود در دیتاست برای هر نمونه را در خود دارد و `Y` هر نمونه نیز، قیمت آن است که در ستون ۸ام دیتاست قرار دارد. به همین منظور تابع `index_to_x_y` را تعریف کرده ایم

در نهایت از این `fold-5` در روش `regression` و `neural_network` استفاده خواهیم کرد.

## رگرسیون :

داده های encode شده برای رگرسیون در نهایت به شکل زیر آمد.

	year	usage	price	brand_تیبا صندوقی دار LX	brand_تیبا صندوقی دار SX	brand_چک اتوماتیک S5	brand_چک دنده ای S5	brand_دنا معمولی	brand_دنا معمولی 1700cc	brand_سند LX	...	body_یک نگه رنگ	loc_1_تهران	loc_1_تهران	loc_1_تهران
0	1397	55000.0	70000000.0	0	0	0	0	0	0	0	...	0	0	0	0
1	1392	40000.0	34000000.0	0	0	0	0	0	0	0	...	1	0	0	0
2	1384	300000.0	36500000.0	0	0	0	0	0	0	0	...	0	0	0	0
3	1386	250.0	27200000.0	0	0	0	0	0	0	0	...	1	0	1	0
4	1387	277.0	30000000.0	0	0	0	0	0	0	0	...	0	0	0	0

در این بخش با استفاده از ۵-fold ی که پیش تر ساخته بودیم ۵ بار مدل رگرسیون میسازیم و هر بار یکی از بخش های داده را برای اعتبار سنجی و بقیه را برای آموزش استفاده میکنیم/

سپس این مدل هارا در لیستی به نام reg\_models نگه می داریم . زیرا که برای رسم نمودار ها در نهایت و مقایسه نتایج به اطلاعات موجود در این مدل ها نیاز داریم مثل اطلاعات مربوط به ضرایب در هر مدل .

در مرحله ی بعدی ،تابع predict را روی مدل رگرسیون خطی برای x\_train و x\_test فراخوانی می کنیم . با این کار پیش بینی های مدل خود را برای این ورودی ها بدست می آوریم و آنها را در y\_hat\_train و y\_hat\_test می ریزیم .

در نهایت برای مقایسه ی پیش بینی های مدل خود و مقادیر واقعی آنها در دیتاست ، از ۲ معیار با نام های mean\_absolute\_error یا همان mae و r2\_score که از sklearn.metrics در ابتدای کار import شده اند ، استفاده می کنیم .در زیر به طور خلاصه هر کدام از این ۲ معیار را توضیح می دهیم :

- mean\_absolute\_error :

برای محاسبه ی خطای مدل ها ، وقتی با داده های عددی سر و کار داریم ، یکی از متداول ترین روش ها ،محاسبه ی mae است . این روش بسیار بدیهی و ساده است و کاری که انجام می دهد این است که قدرمطلق اختلاف های مقادیر پیش بینی شده توسط مدل و مقدار واقعی y در دیتاست ، برای تمام نمونه ها بدست می آورد .سپس از آنها میانگین می گیرد . در زیر فرمول آن را مشاهده می کنید :

$$MAE = \frac{1}{n} \sum_{i=1}^n |f_i - y_i|$$

- r2\_score :

به  $R^2$  ضریب تعیین یا ضریب تشخیص نیز گفته می شود.

تعریف ضریب تعیین  $R^2$  نسبتاً ساده است، این ضریب نشان میدهد که چند درصد تغییرات متغیر وابسته به وسیله متغیر مستقل تبیین می شود یا به عبارت دیگر ضریب تعیین نشان دهنده این است که چه مقدار از تغییرات متغیر وابسته تحت تاثیر متغیر مستقل مربوطه بوده و مابقی تغییرات متغیر وابسته مربوط به سایر عوامل میباشد.

$R\text{-squared} = \text{Explained variation} / \text{Total variation}$  ضریب تعیین همیشه بین ۰ و ۱۰۰٪ است:

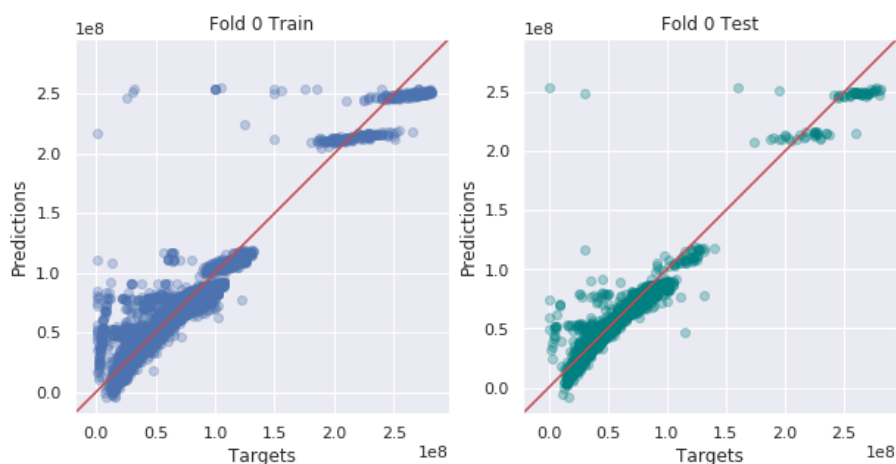
۰٪ نشان می دهد که مدل هیچ یک از تغییرپذیری داده های پاسخ در اطراف میانگین آن را تبیین نمی کند.

۱۰۰٪ نشان می دهد که مدل همه تغییرپذیری داده های پاسخ در اطراف میانگین آن را تبیین می کند.

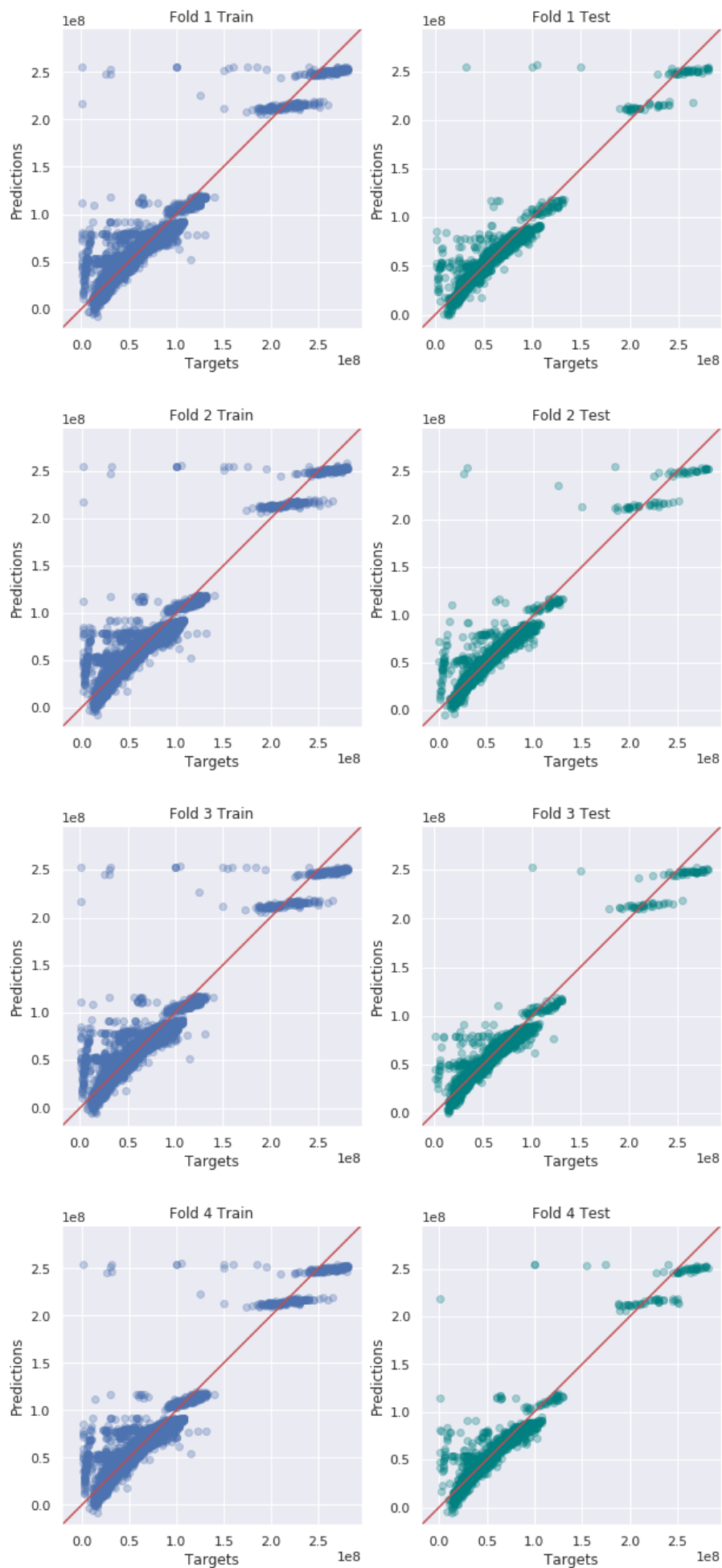
علاوه بر لیست مدل ها که در بالا ذکر شد، تمامی مقادیر پیش بینی مدل ها، اطلاعات مربوط به  $r2\_score$  و  $mae$  آنها، و حتی مقادیر  $y$  موجود در دیتاست، همه ی این اطلاعات هم در بخش **train** و هم در بخش **test**، در لیست های جداگانه با نام های مرتبط ریخته شده و نگه داری می شود. چرا که بعضی برای مقایسه و نتیجه گیری ها و بعضی برای رسم نمودار و مساله های دیگر در بخش های مختلف کار، مانند ساخت **Gui** استفاده شده اند.

در اینجا، نگاهی به این ۵ مدل می اندازیم و نمودار های مربوط به آنها را مشاهده می کنیم که نشان می دهند در هر کدام از این مدل ها، خط رگرسیون به چه صورت بوده است. محور افقی در این نمودار ها، نشان دهنده ی مقدار قیمت واقعی هر نمونه بوده و محور عمودی مقدار پیش بینی شده ی قیمت ها در این مدل ها را نشان می دهد.

همانطور که مشخص است، نمودار های سمت چپ برای داده ها در بخش **train** و نمودار های سمت راست برای داده ها در بخش **test** نمایش داده شده اند. همین طور پررنگ یا کم رنگ بودن نمودار در بخش های مختلف نشان دهنده ی تراکم داده ها در آن قسمت است :







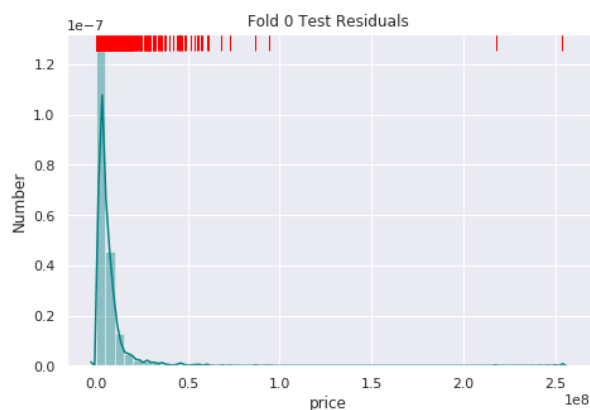
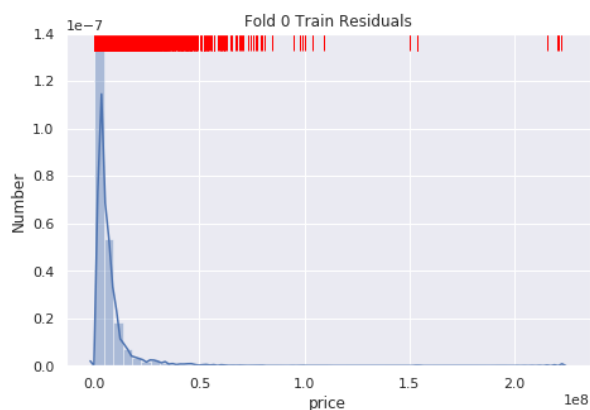
حال نوبت به آن رسیده که از بین این ۵ مدل که با fold های متفاوت بدست آورده ایم ،بهترین آنها را انتخاب کرده و اطلاعات مربوط به ضرایب و عرض از مبدا آن ها را ذخیره میکنیم تا در رابط کاربری از آن ها استفاده کنیم . در اینجا fold چهارم ، نسبت به دیگر fold ها نتایج بهتری داشته است . بنابراین لیست ضرایب مربوط به این مدل و عرض از مبدا آن را بدست می آوریم و در نهایت برای پیش بینی مدل رگرسیون خطی در این تحقیق در قسمت back از Gui از آنها استفاده می کنیم .

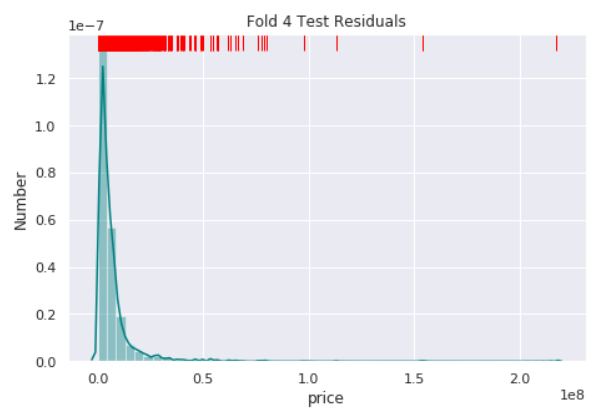
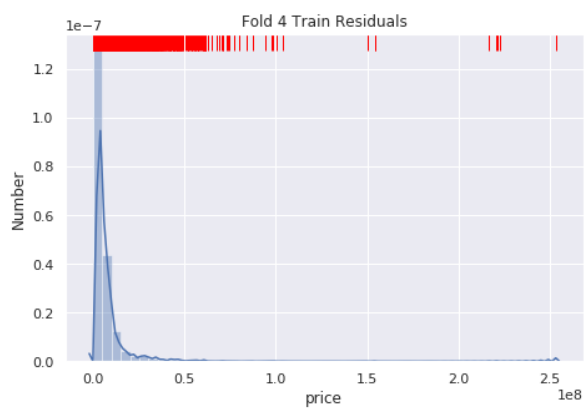
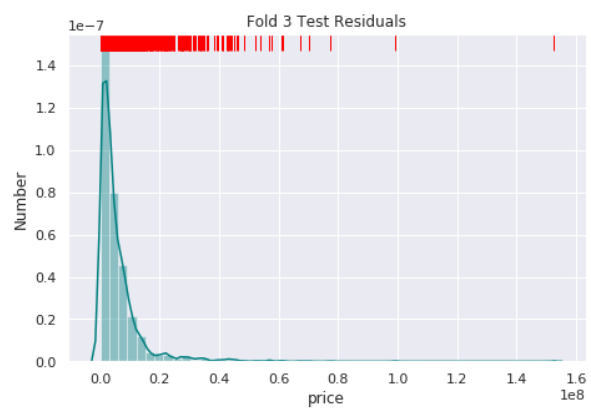
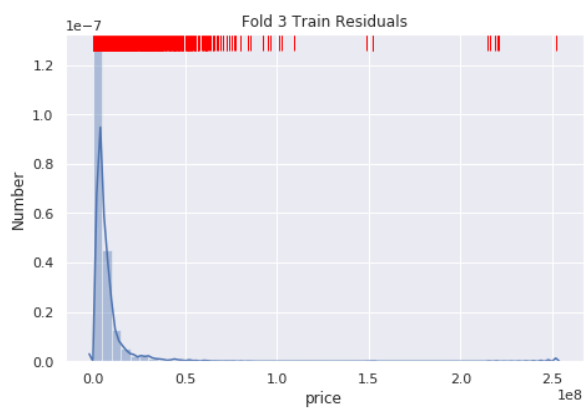
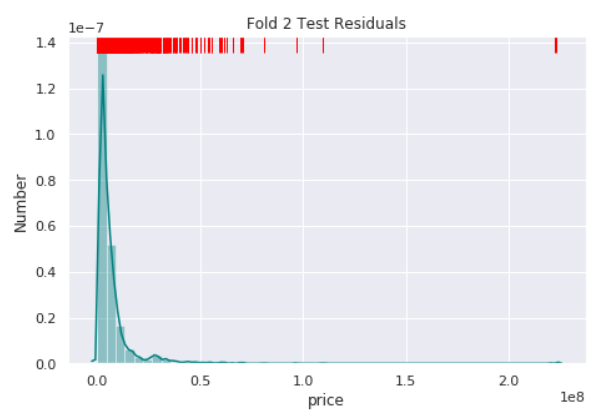
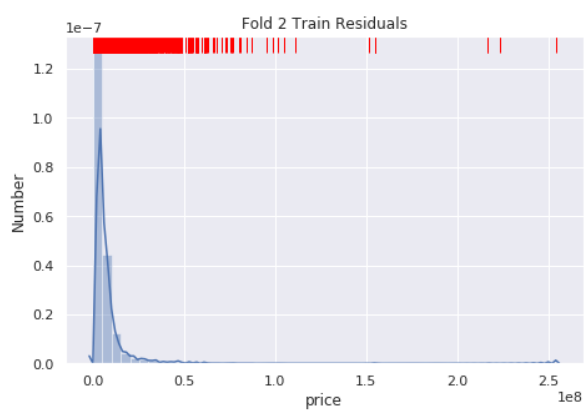
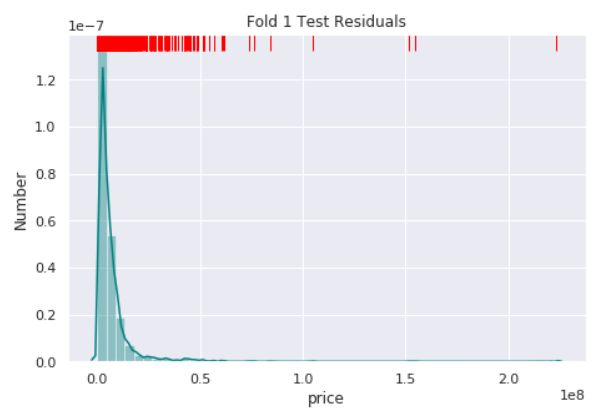
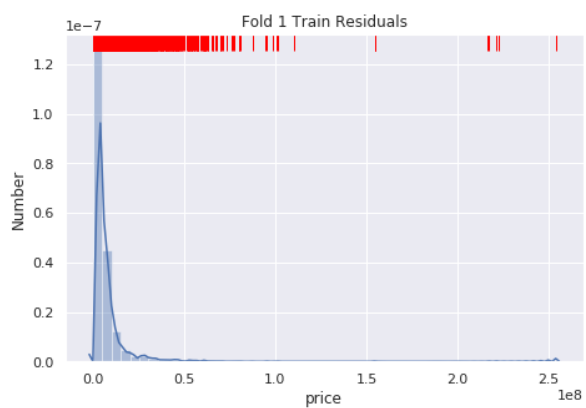
ناگفته نماند که استفاده از k-fold به خاطر اینکه در عملیات preprocessing ، همه دیتاست خود را shuffle کردیم ، چندان نتایج بدست آمده را تغییر نداد . ولی استفاده از آن معمول توصیه می شود برای اینکه اطمینان داشته باشیم که فرآیند یادگیری با هر روشی که هست ، به طور کامل و با رعایت توازن بین داده ها صورت گرفته است . همچنین روش رگرسیون خطی ، روشی ساده است و قطعا نمی توان برای دیتا هایی که خودشان به صورت کامل خطی نیستند که معمولا هم در واقعیت همینطور است انتظار داشت که خیلی دقیق عمل کند . با این حال نتایج نهایی در این دیتاست چندان هم بد نیست و برای یک مدل رگرسیون خطی خیلی هم خوب به نظر می آید .

به عنوان نتیجه نهایی مدل رگرسیون خطی در این تحقیق میانگین این ۵ مدل را با نتایجی که در عکس زیر نمایش داده شده است ، کار این بخش از تحقیق را به پایان میبریم .

```
Regression
mean of mae's for 5-fold on training records = 6.00 milion
mean of mae's for 5-fold on test records = 6.05 milion
mean of r2_score's for 5-fold on training records = 0.918976926609542
mean of r2_score's for 5-fold on test records = 0.91711850542818
```

در زیر نمودار مربوط به خطای این ۵ مدل را مشاهده میکنید.





این نمودار ها در واقع توزیع فراوانی خطاها، یعنی قدر مطلق اختلاف بین مقدار پیش بینی شده توسط مدل و مقدار واقعی آن را نشان می دهند . همانطور که می بینید در این مدل ها ،بیشتر توزیع فراوانی خطا ها ، در حدود صفر تا ده میلیون است و به ندرت خطا های بسیار زیاد دیده شده اند و میانگین خطاها به طور کلی روی `train` و `test` همانطور که در بالا دیدیم ، حدود ۶ میلیون تومان بوده است که برای یک مدل رگرسیون خطی بسیار خوب محسوب می شود . حال به سراغ مدل شبکه عصبی می رویم تا بتوانیم خطاها را از این مقدار هم پایین تر برده و پیش بینی های دقیق تری داشته باشیم .

## مدل شبکه عصبی مصنوعی :

داده های `encode` شده برای مدل شبکه عصبی که در بخش های قبل درباره آن توضیح داده شد ، به صورت زیر هستند :

	brand	color	gear	year	body	loc	usage	price
0	26	6	1	21	0	6	55000.0	70000000.0
1	12	0	1	16	3	5	40000.0	34000000.0
2	24	23	1	8	2	5	300000.0	36500000.0
3	19	15	1	10	3	2	250.0	27200000.0
4	19	10	1	11	2	7	277.0	30000000.0

نکته ی قابل توجه در اینجا ، `scale` کردن این داده های `encode` شده است . `Scale` کردن داده ها به این معناست که همه ی داده های عددی مورد نظر را ، به یک بازه مشخص محدود کند تا داده ها همگی در یک `range` عددی باشند و استاندارد شوند تا در فرآیند یادگیری مشکلی پیش نیاید . برای درک بهتر دستور `scale` کردن ،مثلا در همین دیتاست ، مقادیر `encode` شده به ستون `gear` ، فقط ۱ یا ۲ می باشند در حالی که در ستون `brand` ،مقادیر `encode` شده بسیار بیشتر هستند و شاید از ۱ تا ۴۰ باشند . برای اینکه در فرآیند یادگیری اینگونه تفاوت های `range` های عددی برای ستون های مختلف ، باعث خطا در یادگیری نشوند ، بهتر است دیتاست خود را `scale` کنیم .

در اینجا نیز با اجرای دستور زیر ، همه ی ستون های دیتاست به جز ستون price را scale می کنیم :

```
scaler.fit_transform(neu_encoded[['brand','color','gear','year','body','loc','usage']])
```

حال دیتاست به صورت زیر ، تغییر کرده است :

	brand	color	gear	year	body	loc	usage	price
0	0.926539	-1.616113	0.148077	1.066514	-0.962068	0.690025	-0.361548	70000000.0
1	-0.981747	-2.658989	0.148077	0.131907	1.731122	0.298753	-0.512325	34000000.0
2	0.653927	1.338703	0.148077	-1.363464	0.833392	0.298753	2.101139	36500000.0
3	-0.027604	-0.051798	0.148077	-0.989622	1.731122	-0.875064	-0.911884	27200000.0
4	-0.027604	-0.920862	0.148077	-0.802700	0.833392	1.081297	-0.911612	30000000.0

حالا می خواهیم بررسی کنیم که اگر روی دیتاست خود ، الگوریتم pca که از sklearn.decomposition import کرده ایم ، پیاده سازی کنیم ، نتایج بهتری بدست می آوریم یا نه . ابتدا به توضیح مختصری درباره الگوریتم pca می پردازیم .

## PCA (Principal Component Analysis)

یکی از کاربردهای اصلی PCA در عملیات کاهش ویژگی Dimensionality Reduction است. PCA همانطور که از نامش پیداست میتواند مولفه های اصلی را شناسایی کند و به ما کمک می کند تا به جای اینکه تمامی ویژگی ها را مورد بررسی قرار دهیم، یک سری ویژگی هایی را که ارزش بیشتری دارند، تحلیل کنیم. در واقع PCA آن ویژگی هایی را که ارزش بیشتری فراهم می کنند برای ما استخراج می کند.

اجازه بدهید با یک مثال شروع کنیم. فرض کنید یک فروشگاه میخواهد ببیند که رفتار مشتریان در خرید یک محصول خاص مثلا یک کفش خاص چگونه بوده است. این فروشگاه، اطلاعات زیادی از هر فرد دارد (همان ویژگیهای آن فرد). برای مثال این فروشگاه، از هر مشتری ویژگی های زیر را جمع آوری کرده است:

سن، قد، جنسیت، محل تولد شخص ( غرب ایران، شمال ایران، شرق ایران یا جنوب ایران )، میانگین تعداد افراد خانواده، میانگین درآمد، اتومبیل شخصی دارد یا خیر و در نهایت اینکه این شخص بعد از بازدید کفش خریده است یا خیر .  
۷ ویژگی اول ابعاد مسئله ما را می ساختند و ویژگی آخر هدف (target) می باشد.

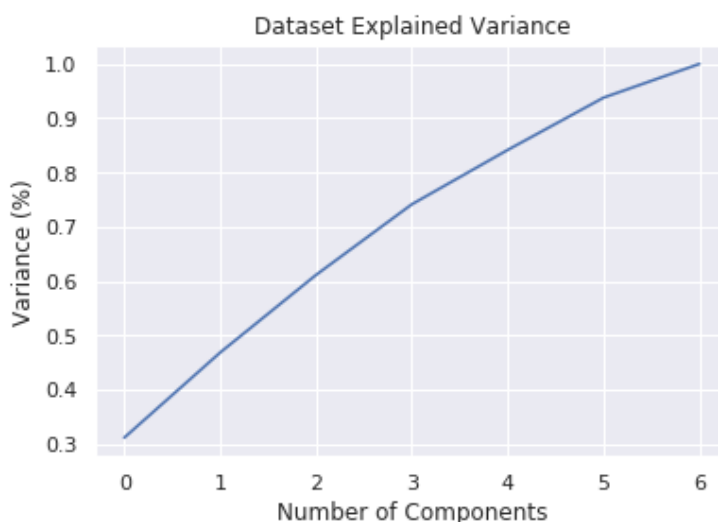
ویژگی های بالا را میتوان برای مجموعه داده ، در ۷ بعد رسم کرد. حال به PCA بازمی گردیم. PCA میتواند آن مولفه هایی را انتخاب کند که نقش مهمتری در خرید دارند. برای مثال، مدیر فروش به ما گفته است که به جای اینکه هر ۷ بعد را در تصمیم گیری دخالت دهیم، نیاز به ۳ بعد (۳ ویژگی) داریم تا بتوانیم آنها را بر روی یک رابط گرافیکی ۳ بعدی به نمایش دریاوریم. پس در واقع نیاز داریم ۷ بعد را به ۳ بعد کاهش دهیم. به این کار در اصطلاح کاهش ابعاد داده Dimensionality Reduction می گویند. PCA می تواند این کار را برای ما انجام دهد. PCA با توجه به داده ها و دامنه ی تغییرات هر کدام از آن ها، میتواند ویژگی هایی را انتخاب کند که تاثیر حداکثری در نتیجه نهایی داشته باشند. در مثال بالا (فروشگاه)، فرض کنید ویژگی قد، تاثیر زیادی در اینکه یک فرد از فروشگاه خرید کند نداشته باشد. PCA این قضیه را متوجه می شود و در الگوریتم خود ویژگی قد را تا جای ممکن حذف می کند . در واقع در فرآیند تبدیل ۷ ویژگی به ۳ ویژگی نهایی (که با توجه به درخواست مدیر فروش به دنبال آن هستیم) ، PCA ویژگی قد را کمتر دخالت می دهد. اینگونه است که ویژگی های مهمتر از نظر PCA وزن بیشتری در تولید ویژگی های کاهش یافته پیدا می کنند.

البته این بدان معنا نیست که در فرآیند کاهش ابعاد داده ، PCA دقیقا همان ویژگی ها را حذف می کند . بلکه PCA توان این را دارد که به یک سری ویژگی جدید برسد. مثلا این الگوریتم ممکن است به این نتیجه برسد که افرادی که در شمال و غرب ایران زندگی می کنند و سن آنها بالای ۴۰ سال است ، احتمال خرید بالایی دارند در حالی که برعکس این قضیه احتمال خرید را بسیار کمتر می کند . در واقع اینجا PCA به یک ویژگی ترکیبی از محل تولد شخص و سن او رسیده است. این دقیقا یکی از قدرت های الگوریتم PCA در کار بر روی داده ها است

خب ، حالا که متوجه مفهوم و کاربرد pca شدیم ، می خواهیم بررسی کنیم که آیا pca در اینجا می تواند به ما کمک کند . برای اینکار با دستور زیر ، pca را روی دیتاست خود اجرا می کنیم :

```
pca = PCA().fit(neu_encoded.drop('price',axis=1))
```

سپس نمودار فراوانی تجمعی `explained_variance_ratio` را رسم میکنیم.



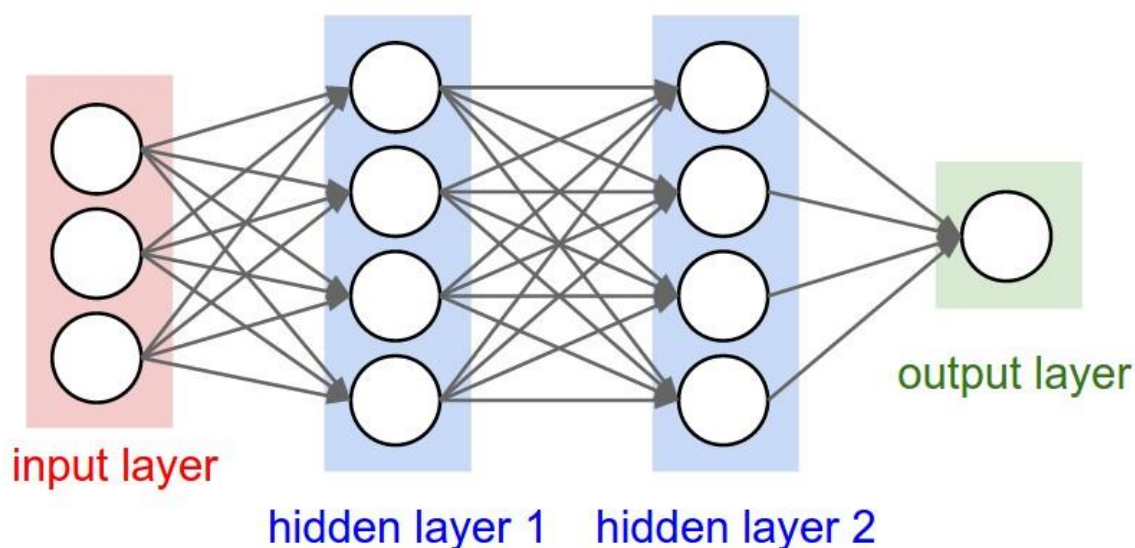
با توجه به این نمودار ، متوجه می شویم که **pca** در اینجا کاربردی برای ما نخواهد داشت ، پس بهتر است از آن استفاده نکنیم . چرا که اگر **pca** می توانست به ما در اینجا کمک کند ، این نمودار نباید اکیدا صعودی می بود ، و می بایست در نقطه ای به صورت تقریبا افقی در می آمد تا به ما نشان می داد که از آن نقطه به بعد ، ویژگی های بعدی ، اطلاعات زیادی به ما اضافه نمی کنند و می توان ابعاد مساله را کاهش داد . اما همانطور که می بینیم این اتفاق نیافتاده و این نشان می دهد که تمامی ویژگی های موجود به ما کمک می کنند و تاثیر گذار در مقدار پیش بینی مدل هستند . پس نمی توان ابعاد را کاهش داد و چیزی را نادیده گرفت .

حال زمان این رسیده است که مدل شبکه عصبی مورد نظرمان را بسازیم.

انتخاب ساختار مدل شبکه عصبی :

در شبکه های عصبی بستر مناسبی برای ایجاد انواع ساختار های مختلف فراهم می باشد که هر کدام از این ساختار ها مخصوص نوع خاصی از تحلیل داده می باشند ، مثلا شبکه های **convolutional** مخصوص کار روی داده های تصویری می باشد ، یا برای مثال شبکه هایی که با ساختار **RNN** دارای حافظه ساخته میشوند بیشتر برای تولید متن های معنی دار طولانی استفاده میشود که در آن ها نیاز است اطلاعات کلمه های قبل را نیز به عنوان ورودی گرفت تا بتوان جمله های معنی دار ساخت. یکی از انواع شبکه هایی که برای کار با داده های عددی بسیار مناسب است شبکه هایی هستند که از لایه های **Dense** تشکیل شده اند. شبکه هایی که از لایه های **Dense** تشکیل شده اند

را اصطلاحاً fully-connected مینامند و دلیل این نام گذاری این است که در این نوع از شبکه ها تمام نورون های هر لایه با تمام نورون های لایه های قبل و بعدشان به وسیله یک یال وصل شده اند و از این طریق میتوانند اطلاعات را از لایه ورودی به لایه خروجی منتقل کنند در زیر تصویری از یک شبکه fully-connected قرار دارد.



این نوع شبکه ها همان طور که در تصویر نیز واضح است یک لایه ورودی و یک لایه خروجی دارند و تعدادی لایه نیز به عنوان لایه های پنهان بین این دو لایه قرار دارند.

از آن جا که تعداد لایه های استفاده شده در شبکه عصبی و همینطور تعداد نورون های استفاده شده در هر لایه تاثیر بسزایی روی نتیجه به دست آمده از شبکه ما خواهد داشت نیاز است که اطلاعات این چینی را به بهترین شکل ممکن انتخاب کنیم. لازم به ذکر است که هیچ راه صد در صدی برای انتخاب بهترین ساختار شبکه برای مسئله وجود ندارد و تحلیل گران داده تنها با آزمون و خطا و عوض کردن مقادیر مختلف ویژگی های شبکه عصبی و بررسی عملکرد شبکه عصبی به دست آمده، بهترین شبکه ای که به آن دست پیدا کرده اند را برای استفاده انتخاب میکنند.

برای ایجاد شبکه های عصبی مختلف و تست کردن آن ها نیاز است که ما این شبکه های عصبی را ابتدا روی داده های آموزشی Train کنیم و سپس عملکرد آن را روی داده های Test و بررسی کنیم.

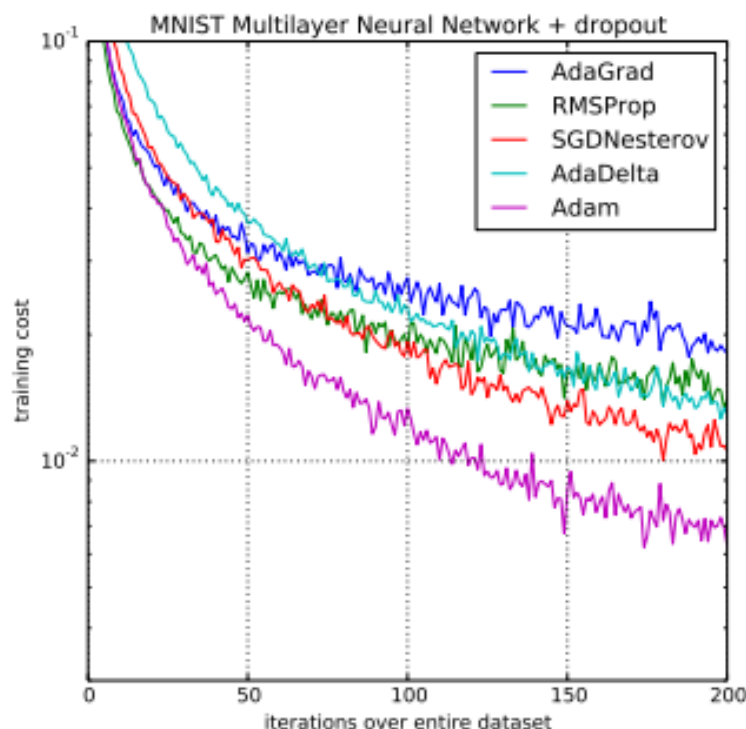


به دلیل اینکه تعداد داده های نسبتاً زیاد است و آموزش شبکه عصبی روی کل این داده ها زمان زیادی طول میکشد ما بجای این که این کار را روی کل داده ها انجام دهیم یکبار داده ها را shuffle میکنیم و سپس یک sample از اکل داده ها انتخاب میکنیم که تنها شامل ۰.۳٪ از کل داده ها است یعنی بجای کار روی ۱۱۳۷۲ داده این کار را روی ۳۴۱۲ داد انجام میدهیم که باعث سریع تر شدن فرایند یادگیری و بررسی عملکرد مدل میشود. لازم به ذکر است که ما فقط برای پیدا کردن مدل مورد نظر این کار را انجام میدهیم و در نهایت برای آموزش نهایی مدل مورد نظر از تمام داده ها استفاده میکنیم که باعث میشود دقت شبکه ما افزایش یابد.

همچنین در اینجا برای آموزش مدل خود از ۵۰ epoch استفاده میکنیم اما در آموزش اصلی از ۲۰۰ epoch استفاده میکنیم، پس توجه کنید که برخی از این تنظیمات فقط برای پیدا کردن مدل مناسبمان است.

از آن جا که تعداد داده ها در آموزش مدل اصلی بیشتر خواهد بود، ما در این دنبال مدلی هستیم که قدرت آن از این sample ای که انتخاب کرده ایم بیشتر باشد، و این امر با شکستگی های موجود در نمودار epoch – loss مشخص میشود که در زمان مشاهده این موضوع بیشتر راجع به آن صحبت میکنیم.

برای optimize کردن مدل شبکه عصبی خود از optimizer، Adam استفاده میکنیم که یکی از قوی ترین optimizer ها میباشد و از سرعت و دقت بهتری نسبت به بسیاری دیگر از optimizer ها برخوردار است



برای مثال در تصویر بالا عملکرد چند Optimizer مختلف روی یک دیتاست معروف در مبحث شبکه های عصبی را مشاهده میکنید که در آن Adam از بقیه Optimizer ها عملکرد بسیار بهتری از خود نشان داده است و معمولاً جزو اصلی ترین انتخاب ها میباشد.

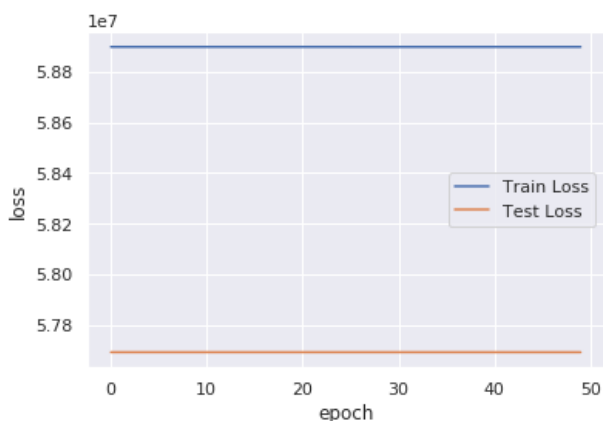
برای محاسبه مقدار Loss در شبکه عصبی نیز از mean\_absolute\_error یا میانگین قدر مطلق خطا ها استفاده میکنیم که در این فاکتور پیش بینی های انجام شده با مقادیر صحیح خروجی مقایسه میشوند و از هم تفریق میشوند. نتیجه این کار لیستی از خطا ها برای پیش بینی ما است که با میانگین گرفتن از این لیست به mae میرسیم که هدف ما نیز در این پروژه کم کردن مقدار این خطا است.

### پیدا کردن شبکه عصبی مورد نظر :

در ابتدا برای ساختن شبکه عصبی نیاز است که ۷ نورون در لایه ورودی قرار دهیم زیرا سایز ورودی مسئله ما ۷ است که شامل brand – color – year – gear – body – location – usage است و خروجی ما نیز قرار است یک عدد باشد پس برای لایه خروجی از یک نورون استفاده میکنیم.

معمولاً برای ساختن شبکه عصبی مناسب مسئله مان از شبکه های بسیار ساده شروع میکنیم و کم کم آن را پیچیده تر میکنیم و تعداد نورون ها و لایه ها را افزایش میدهیم و تغییرات عملکرد شبکه عصبی خود را بررسی میکنیم و بهترین را انتخاب میکنیم.

مدل ۱ : در این مدل از یک لایه پنهان با ۲ نورون استفاده کرده ایم که در زیر نمودار تغییرات مقدار خطا بر حسب epoch را مشاهده میکنید.

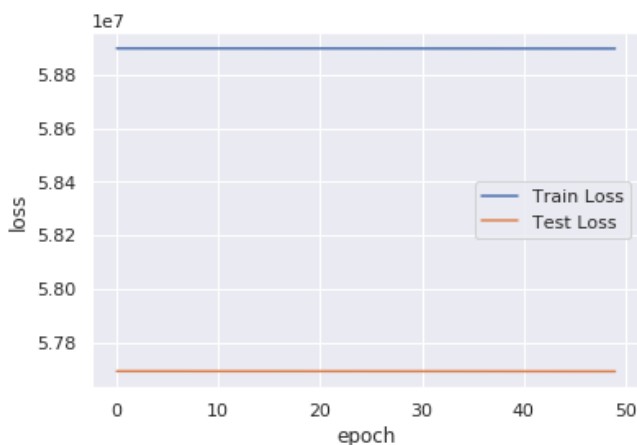
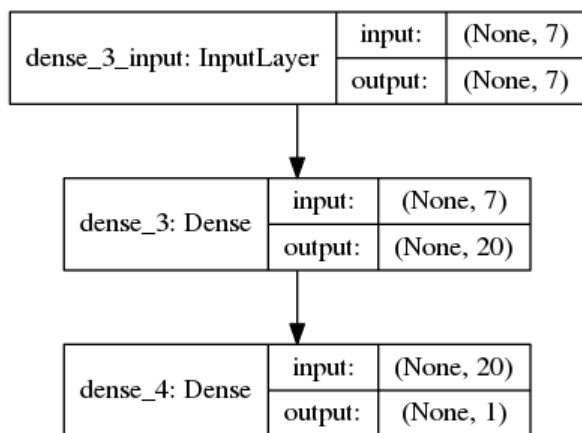


بهترین مقدار خطا روی داده Train و Test نیز به شکل زیر است.

Best Loss = 58898115.06911636  
Best Validation Loss = 57692162.955595024

مقدار خطا بسیار بالا است و همینطور که از نمودار مشخص است مدل ما توانایی یادگیری این داده ها را نداشته است و دقت آن پیشرفتی نداشته است پس تعداد نورون های این لایه را بیشتر میکنیم.

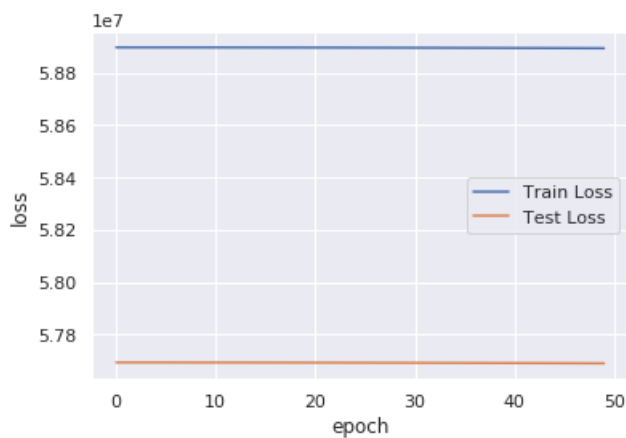
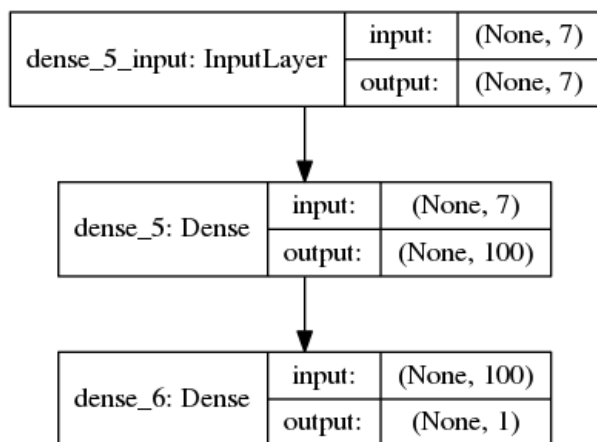
مدل ۲: یک لایه پنهان با ۲۰ نورون



Best Loss = 58897471.25809274  
Best Validation Loss = 57691523.651865005

پیشرفت خاصی نداشتیم.

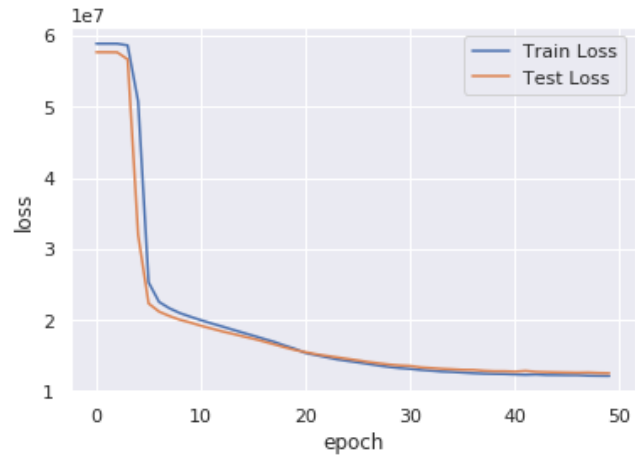
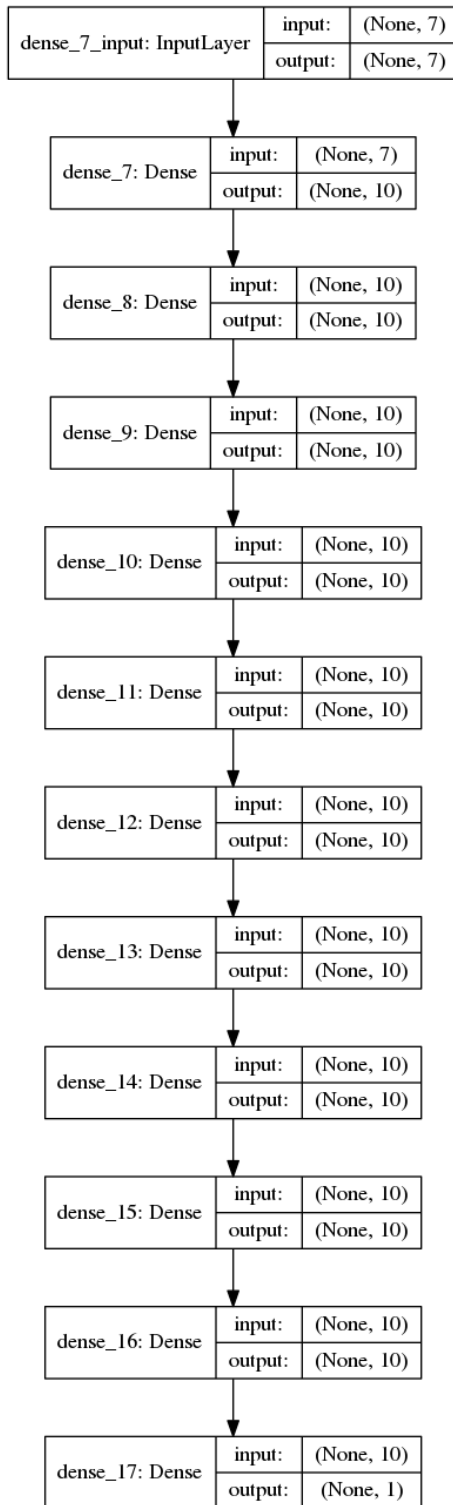
مدل ۳: یک لایه پنهان با ۱۰۰ نورون.



Best Loss = 58894687.68153981  
Best Validation Loss = 57688721.32149281

در این مرحله ما با استفاده از ۱۰۰ نورون در یک لایه هم نتوانستیم بهبودی در عملکرد مدل شاهد باشیم. حال همین ۱۰۰ نورون را به صورت ۱۰ لایه با ۱۰ نورون در هر لایه پشت سر هم قرار می‌دهیم تا عملکرد آن را بررسی کنیم

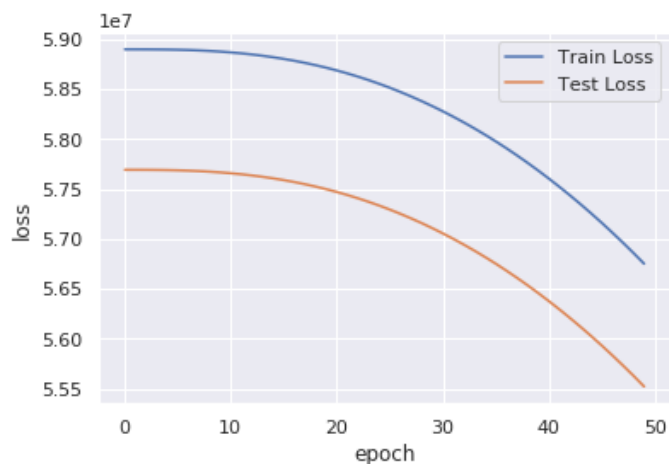
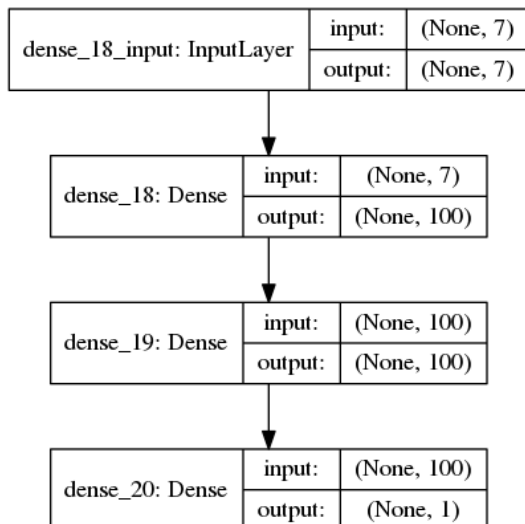
مدل ۴ : ۱۰ لایه با ۱۰ نورون در هر کدام



**Best Loss = 12136552.392825896**  
**Best Validation Loss = 12514539.126110123**

همان طور که میبینید مدل ما شروع به یادگیری کرد و این تغییر شکل چینش نورون ها تاثیر بسزایی در بهبود کیفیت آنداشت ، و همانطور که مشاهده میکنید میانگین خطای مدل ما از ۵۷ میلیون به ۱۲ میلیون کاهش پیدا کرد. پس از این جا به بعد مدل یک لایه ای خود را در جهت زیاد شدن تعداد لایه ها جلو میبریم و عملکردش را بررسی میکنیم.

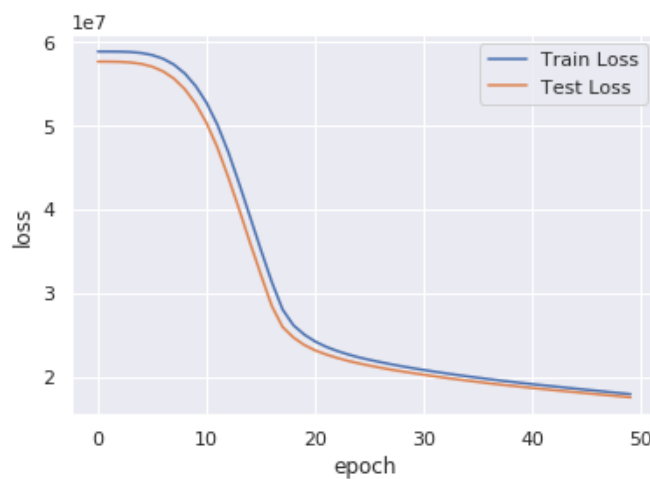
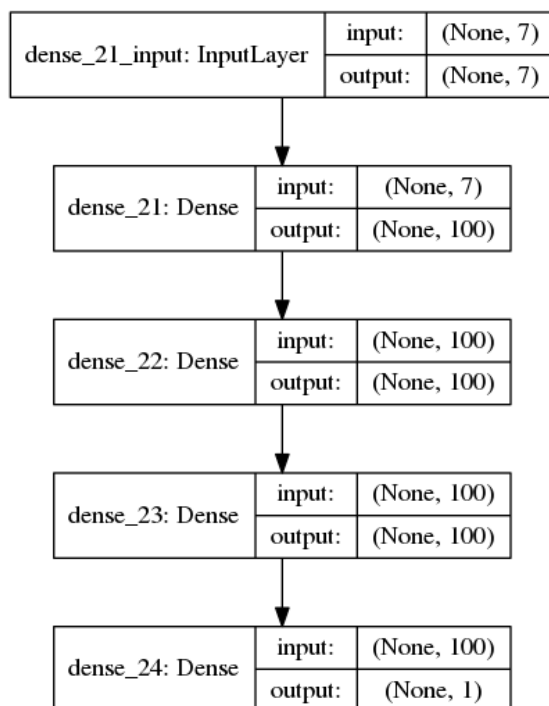
مدل ۵: دو لایه پنهان با ۱۰۰ نورون



Best Loss = 56751270.96062992  
Best Validation Loss = 55521739.76554174

مدل ما شروع به یادگیری کرده است اما هنوز قدرت کافی را ندارد.

مدل ۶: سه لایه پنهان با ۱۰۰ نورون در هر لایه



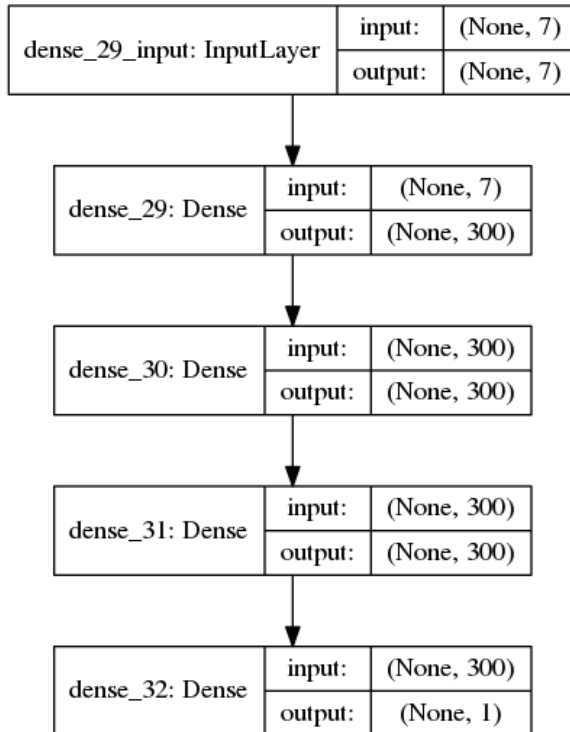
Best Loss = 17930074.817147855  
Best Validation Loss = 17556470.962699823

قدرت مدل ما در حال افزایش است ، حال تاثیر افزایش تعداد نورون های لایه ها را تست میکنیم.

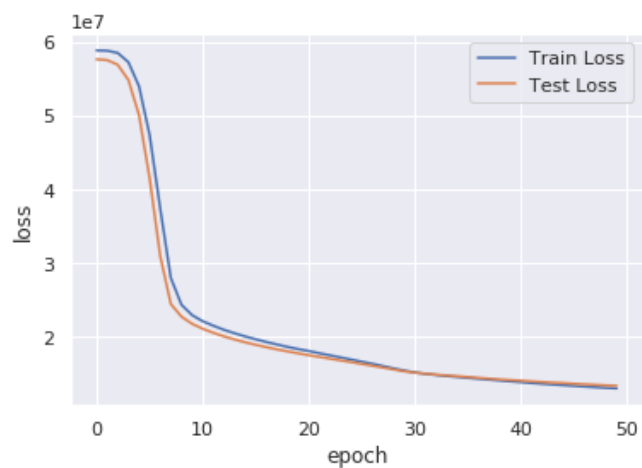
در مدل ۷ سه لایه پنهان با ۲۰۰ نورون در هر لایه را آزمایش میکنیم که نتایج آن به شکل زیر خواهد بود

**Best Loss = 14766540.31671041**  
**Best Validation Loss = 14841652.156305507**

از آن جا که باعث پیشرفت مدل میشود تعداد نورون های لایه ها را به ۳۰۰ میرسانیم



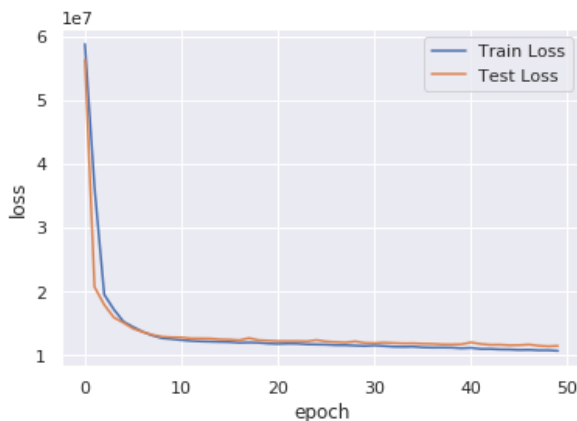
مدل ۸ : سه لایه پنهان به ۳۰۰ نورون در هر لایه



**Best Loss = 13044303.287401576**  
**Best Validation Loss = 13401604.696269982**

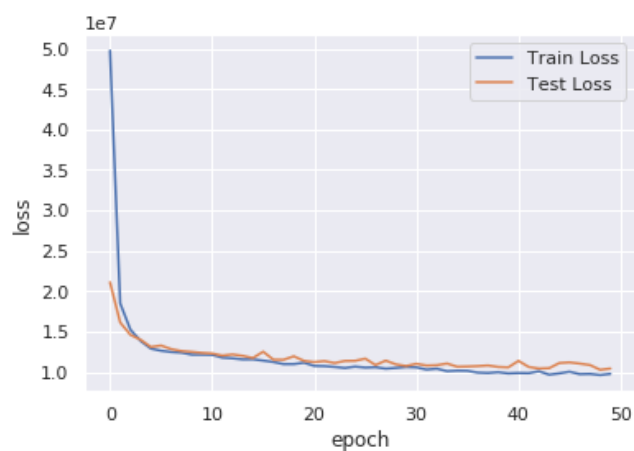
شبکه را عمیق تر میکنیم.

مدل ۹ : ۵ لایه پنهان با ۳۰۰ نورون در هر لایه



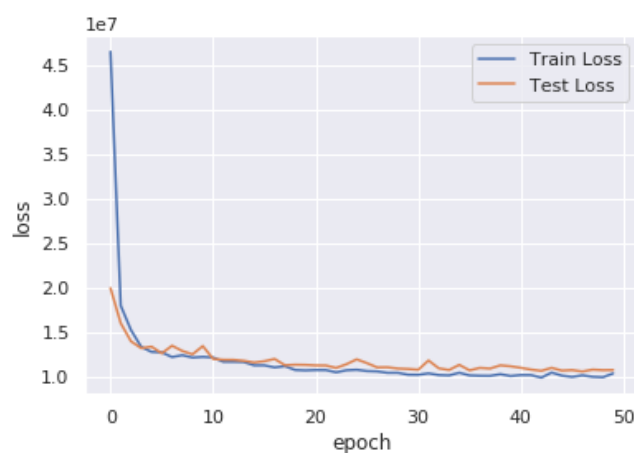
**Best Loss = 10742462.73928259**  
**Best Validation Loss = 11456842.607460035**

در مدل شماره ۱۰ از ۷ لایه پنهان با ۳۰۰ نورون استفاده میکنیم که نتایج آن به شکل زیر میشود.



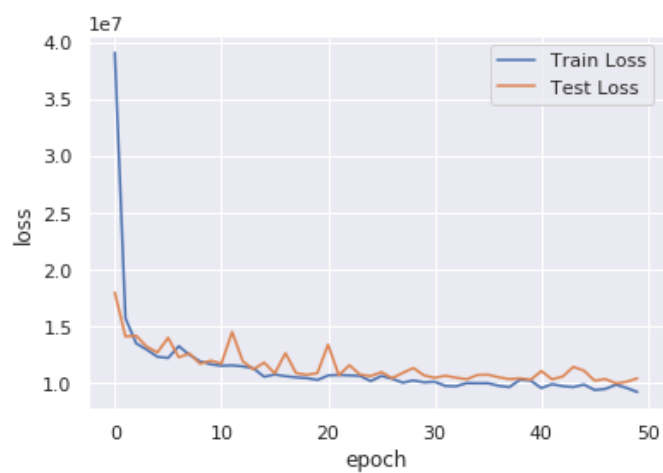
Best Loss = 9637670.657480314  
Best Validation Loss = 10284532.454706928

در مدل ۱۱ تعداد نورون های لایه ها را به ۴۰۰ افزایش میدهیم.



Best Loss = 9854829.588145232  
Best Validation Loss = 10549011.047957372

در مدل ۱۲ باز هم شبکه را عمیق تر میکنیم و از ۹ لایه پنهان با ۴۰۰ نورون در هر لایه استفاده میکنیم



Best Loss = 9240507.233158356  
Best Validation Loss = 9996139.584369449

اگر به نمودار ها دقت کنید متوجه میشوید که با زیاد شدن تعداد لایه ها و نورون ها شکستگی های نمودار ها در حال افزایش است. یکی از دلایل اصلی این شکستگی ها در نمودار ها کم بودن تعداد داده های آموزش میباشد و با آموزش دادن شبکه عصبی مورد نظرمون روی تمام داده ها این مشکل حل خواهد شد و خطای شبکه راحت تر کم میشود. یکی دیگر از عوامل تاثیر گذار نیز این است که قدرت شبکه عصبی که طراحی کرده ایم برای این تعداد داده زیاد است و این باعث شکستگی در این نمودار ها میشود و از آن جا که ما قصد داریم این شبکه را روی داده های بیشتری Train کنیم پس به دنبال ساختاری با قدرت بیشتری هستیم پس شبکه ای را انتخاب میکنیم که شکستگی های روی نمودار آن نه خیلی کم باشد که روی داده اصلی نتیجه خوبی نگیریم. نه خیلی زیاد باشد قدرت مدل ما که روی داده اصلی **overfit** شود و عملکرد خوبی روی داده های جدیدی که برای پیش بینی به آن داده میشود نداشته باشد.

در مدل شماره ۱۳ از ۱۱ لایه پنهان با ۴۰۰ نورون در هر لایه کمک میگیریم و نتایج آن به شکل زیر است.



در مدل شماره ۱۴ نیز تعداد نورون های لایه ها را به ۵۰۰ میرسانیم تا قدرت مدل ما افزایش یابد و در مدل ۱۵ به ۵۵۰ نورون در هر لایه میرسانیم که نتایج مدل ۱۵ به شکل زیر است.





و در نهایت با اندکی تغییر در ساختار این مدل و کم کردن تعداد نورون های لایه ی اول و آخر پنهان به ۳۰۰ به مدل زیر میرسیم. که نتایج آن را در زیر مشاهده میکنید.

dense_101_input: InputLayer	input:	(None, 7)
	output:	(None, 7)

dense_101: Dense	input:	(None, 7)
	output:	(None, 300)

dense_102: Dense	input:	(None, 300)
	output:	(None, 550)

dense_103: Dense	input:	(None, 550)
	output:	(None, 550)

dense_104: Dense	input:	(None, 550)
	output:	(None, 550)

dense_105: Dense	input:	(None, 550)
	output:	(None, 550)

dense_106: Dense	input:	(None, 550)
	output:	(None, 550)

dense_107: Dense	input:	(None, 550)
	output:	(None, 550)

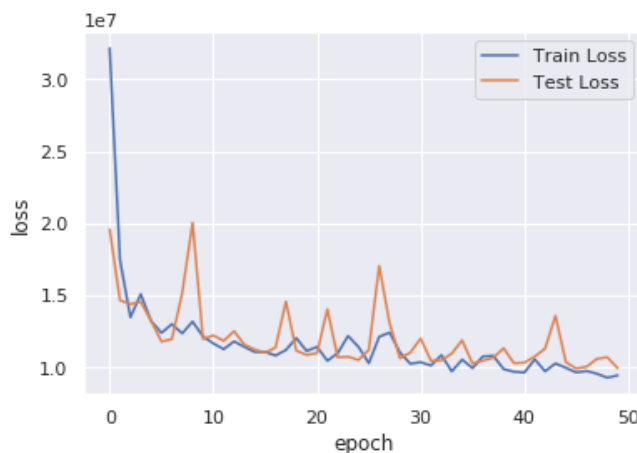
dense_108: Dense	input:	(None, 550)
	output:	(None, 550)

dense_109: Dense	input:	(None, 550)
	output:	(None, 550)

dense_110: Dense	input:	(None, 550)
	output:	(None, 550)

dense_111: Dense	input:	(None, 550)
	output:	(None, 300)

dense_112: Dense	input:	(None, 300)
	output:	(None, 1)



**Best Loss = 9297325.534995625**  
**Best Validation Loss = 9927056.15275311**

اگر به مقادیر خطا توجه کنید میبینید که توانستیم به بهترین مقدار

بین مدل های مختلفی که امتحان کردیم برسیم و همینطور با

بررسی نمودار متوجه میشویم که قدرت مدل ما نیز افزایش یافته

و میتواند الگو های موجود در داده های بیشتری را نیز پیدا کند.

برای بهتر کردن عملکرد مدلما روی داده اصلی همان طور که در

ابتدای این بخش ذکر شد ، تعداد epoch ها را به ۲۰۰ می‌رسانیم.

یکی دیگر از عواملی که برای بهبود عملکرد مدلما استفاده میکنیم

استفاده از kernel\_initializer ، normal است که باعث میشود

مقداردهی اولیه به ضرایب موجود در شبکه عصبیما از توزیعی

نرمال پیروی کند که باعث میشود ضرایب زودتر به مقدار درست برسد

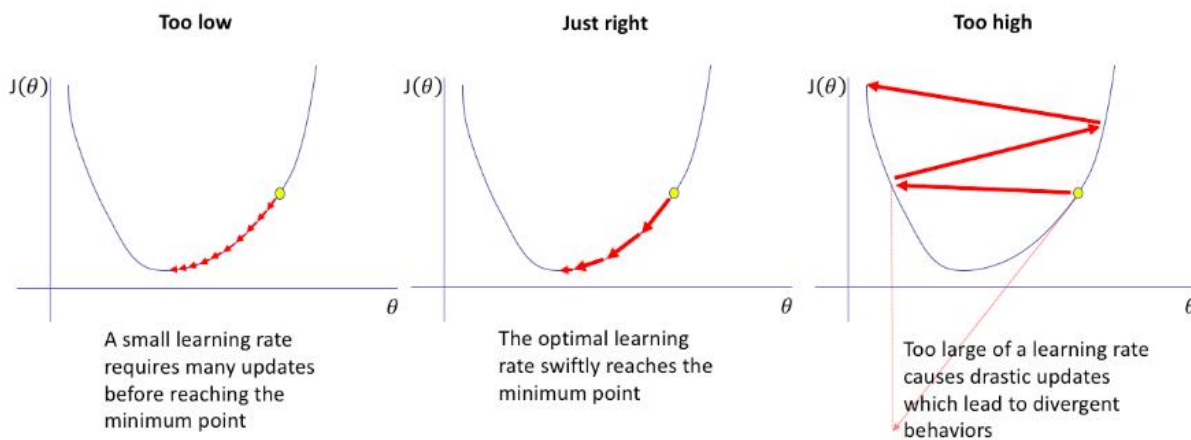
دو عامل دیگر نیز وجود دارند که باعث میشود ما در نهایت به مدل بهتری برسیم که در زیر آن ها را توضیح دادیم.

## Checkpoint

فرآیند یادگیری، می تواند فرآیندی بسیار طولانی باشد و نیاز است تا در مراحل در قسمت های مشخصی از مسیر اطلاعات ذخیره و بازیابی شوند `checkpoint`. دسترسی است که `keras` در اختیار ما می گذارد تا بتوانیم عملیات ذخیره سازی و بازیابی مدل را به راحتی انجام دهیم. در اینجا ما از این دستور استفاده کرده ایم تا در طول فرآیند یادگیری، مقدار `val_loss` نظارت شود و بهترین مقدار آن ذخیره شود. چون `val_loss` میزان خطا را نشان می دهد، باید کمترین مقدار را ذخیره کند. باید به این نکته توجه داشت که ممکن است کمترین خطا، `loss` نهایی نباشد و بنابراین با این کار می توان مدلی که کمترین خطا در طول یادگیری داشته را ذخیره کنیم که بسیار مطلوب است.

## -reduce\_lr :

همیشه در فرآیند های یادگیری ما به دنبال کمترین خطا ها هستیم. در نتیجه همیشه به دنبال نقاط اکسترم هستیم و می دانیم که `learningrate` همان معیاری است که گویی سرعت حرکت به سمت اکسترم های مساله را مشخص می کند. اما باید توجه داشت با اینکه نرخ یادگیری بالا موجب افزایش سرعت شبکه برای رسیدن به حالت بهینه می شود، ممکن است مشکلاتی نیز پیش بیاید. در تصویر زیر شما تفاوت میان نتیجه استفاده چند استراتژی مختلف در انتخاب `Learning Rate` را می بینید:



انتخاب **Lerning rate** های بزرگ ممکن است در گام های اول موجب نزدیک شدن ما به هدف شود، اما در ادامه مدام باعث تلورانس ما در اطراف **Global Optimum** خواهد شد. (شکل سمت راست)

همچنین برعکس، انتخاب **Lr** بسیار کوچک زمان رسیدن به هدف را ممکن است بسیار طولانی کند و عملاً انتخاب مناسبی نخواهد بود. (شکل سمت چپ)

یکی از ایده های حل اشکالات روش های بالا، استفاده از **Lr** پویاست، به طوری که مقدار آن به شکل خطی و یا حتی لگاریتمیک کاهش یابد. تا در ابتدا با گام های بزرگ به سمت جواب حرکت کنیم و رفته رفته با نزدیک تر شدن به جواب مسئله، **Lr** را کاهش دهیم. (شکل وسط)

در اینجا نیز از دستور زیر استفاده شده است

```
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.8, patience=5, min_lr=0.00001)
```

که به این صورت عمل میکند که همواره مقواره خطای داده اعتبار سنجی را در طول epoch ۲۰۰ بررسی میکند و اگر مدل ما در طول epoch ۵ هیچ بهبودی در این زمینه نداشت مقدار **learning rate** به ۰.۸ مقدار قبلی نزول میکند. لازم به ذکر است که **learning rate** اولیه برابر ۰.۰۰۱ است و این کاهش تدریجی **learning rate** تا ۰.۰۰۰۰۱ ادامه پیدا میکند.

پس در اینجا مدل نهایی، یک شبکه عصبی از نوع **Sequential** است که دارای ۲۱ لایه **Dense**، که دو لایه اول و یازدهم دارای ۳۰۰ نورون هستند، لایه های میان این دو لایه، همگی دارای ۵۵۰ نورون بوده و لایه آخر نیز دارای یک نورون است، زیرا در نهایت قرار است یک جواب داشته باشیم. **Activation function** در همه ی این لایه ها و در همه ی نورون ها، تابع **relu** است. **Optimizer** را **adam** انتخاب کرده ایم و در نهایت نوع **loss** را در شبکه عصبی خود، از نوع **mae** قرار داده ایم. به طور کلی تمامی جزئیات مدل خود را، در یک تابع به نام **make\_network** تعریف کرده ایم. چرا که در این صورت لازم نخواهد بود برای تمام ۵ مدل ساخته شده در روش **k\_fold**، هر بار تمام این جزئیات را دوباره بسازیم و می توان با فراخوانی این تابع برای این ۵ مدل خیلی سریع شبکه عصبی هر کدام را ساخت. در زیر می توانید **summary** این مدل شبکه عصبی را ببینید.

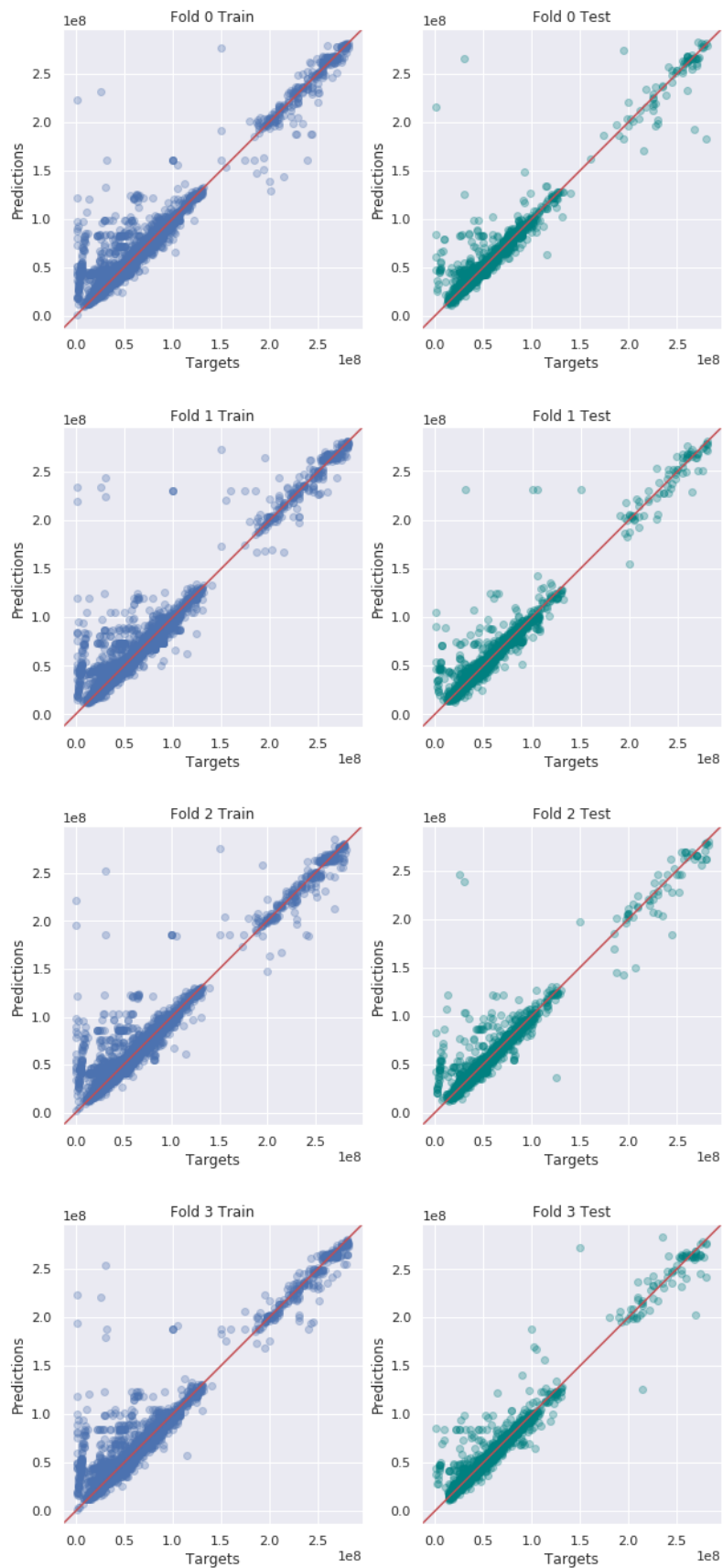
برای **fit** کردن شبکه عصبی، در اینجا نیز مانند مدل رگرسیون خطی، ابتدا **x\_train**, **y\_train**, **x\_test**, **y** را برای هر **fold**، با استفاده از تابع **index\_to\_x\_y**، بدست می آوریم و در لیست هایی با همین نام ها می ریزیم.

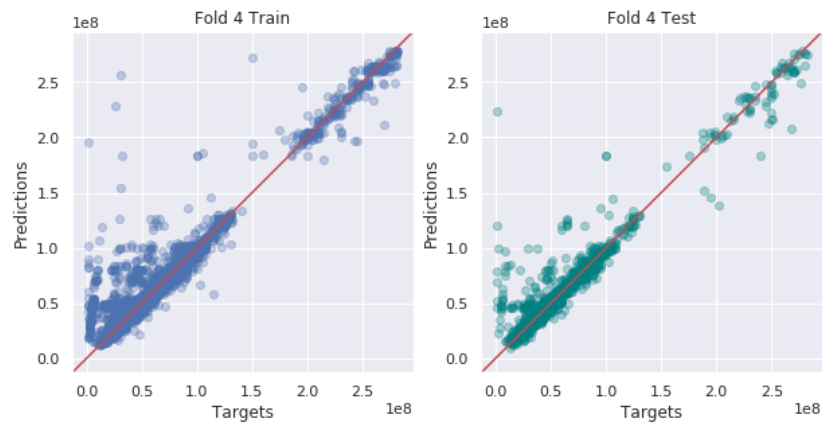
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 300)	2400
dense_2 (Dense)	(None, 550)	165550
dense_3 (Dense)	(None, 550)	303050
dense_4 (Dense)	(None, 550)	303050
dense_5 (Dense)	(None, 550)	303050
dense_6 (Dense)	(None, 550)	303050
dense_7 (Dense)	(None, 550)	303050
dense_8 (Dense)	(None, 550)	303050
dense_9 (Dense)	(None, 550)	303050
dense_10 (Dense)	(None, 550)	303050
dense_11 (Dense)	(None, 300)	165300
dense_12 (Dense)	(None, 1)	301
Total params: 2,757,951		
Trainable params: 2,757,951		
Non-trainable params: 0		

تعاریفی که به صورت مشترک در این بخش و بخش رگرسیون خطی وجود دارند ، اینجا دوباره توضیح داده نشده است . در صورت ابهام در تعاریف ، به تعاریف بخش رگرسیون مراجعه کنید.

دوباره نوبت به نمودار ها برای این ۵ مدل شبکه عصبی می رسد . همانند توضیحاتی که در بخش رگرسیون خطی داده شد ، در اینجا نیز ، محور افقی در این نمودار ها ، نشان دهنده ی مقدار قیمت واقعی هر نمونه بوده و محور عمودی مقدار پیش بینی شده ی قیمت ها در این مدل ها را نشان می دهد.

همانطور که مشخص است ، نمودار های سمت چپ برای داده ها در بخش **train** و نمودار های سمت راست برای داده ها در بخش **test** نمایش داده شده اند . همین طور پررنگ یا کم رنگ بودن نمودار در بخش های مختلف نشان دهنده ی تراکم داده ها در آن قسمت است :

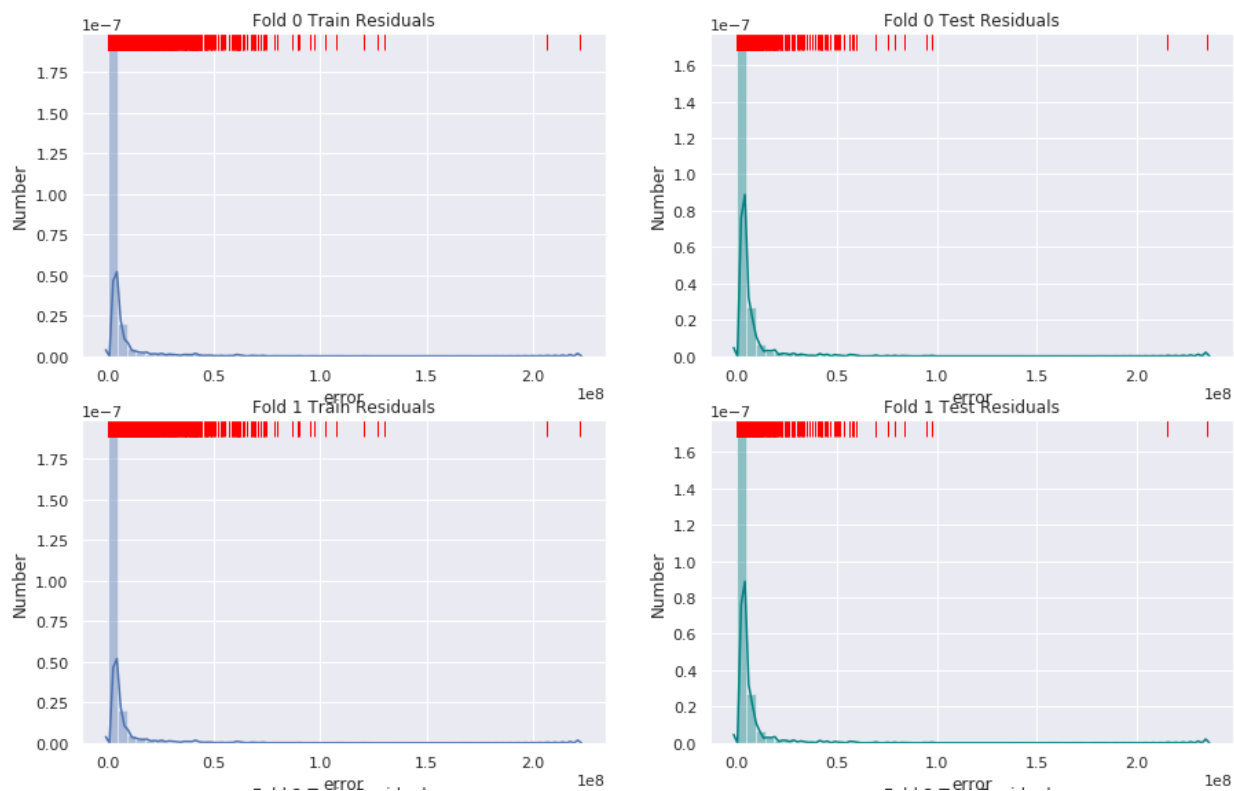


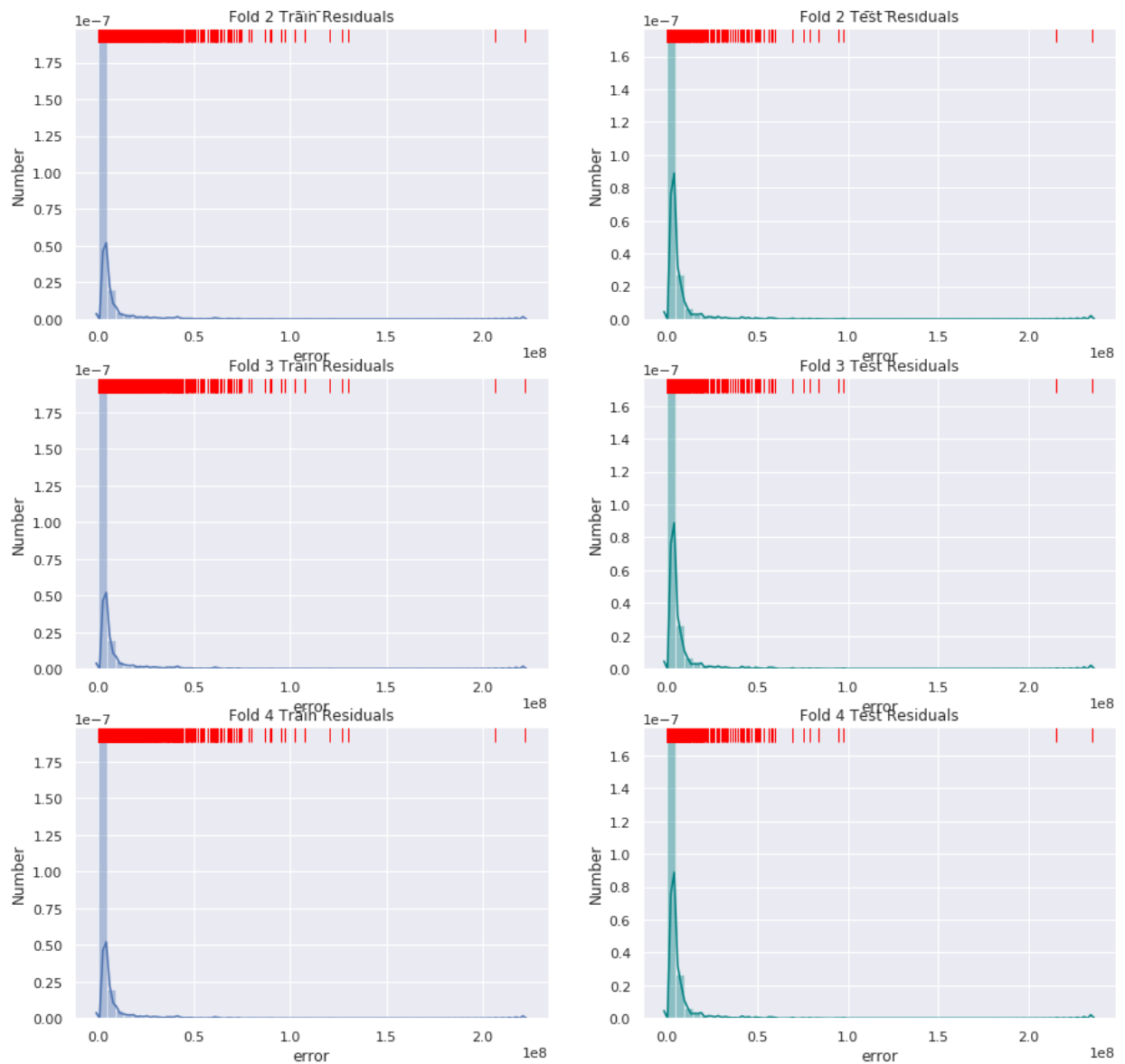


به عنوان نتیجه نهایی مدل شبکه عصبی در این تحقیق ، میانگین خطای این ۵ مدل در تصویر زیر نمایش داده شده است :

```
Neural Network
mean of mae's for 5-fold on training records = 3.54 milion
mean of mae's for 5-fold on test records = 4.52 milion
mean of r2_score's for 5-fold on training records = 0.9435930666367216
mean of r2_score's for 5-fold on test records = 0.928775635706355
```

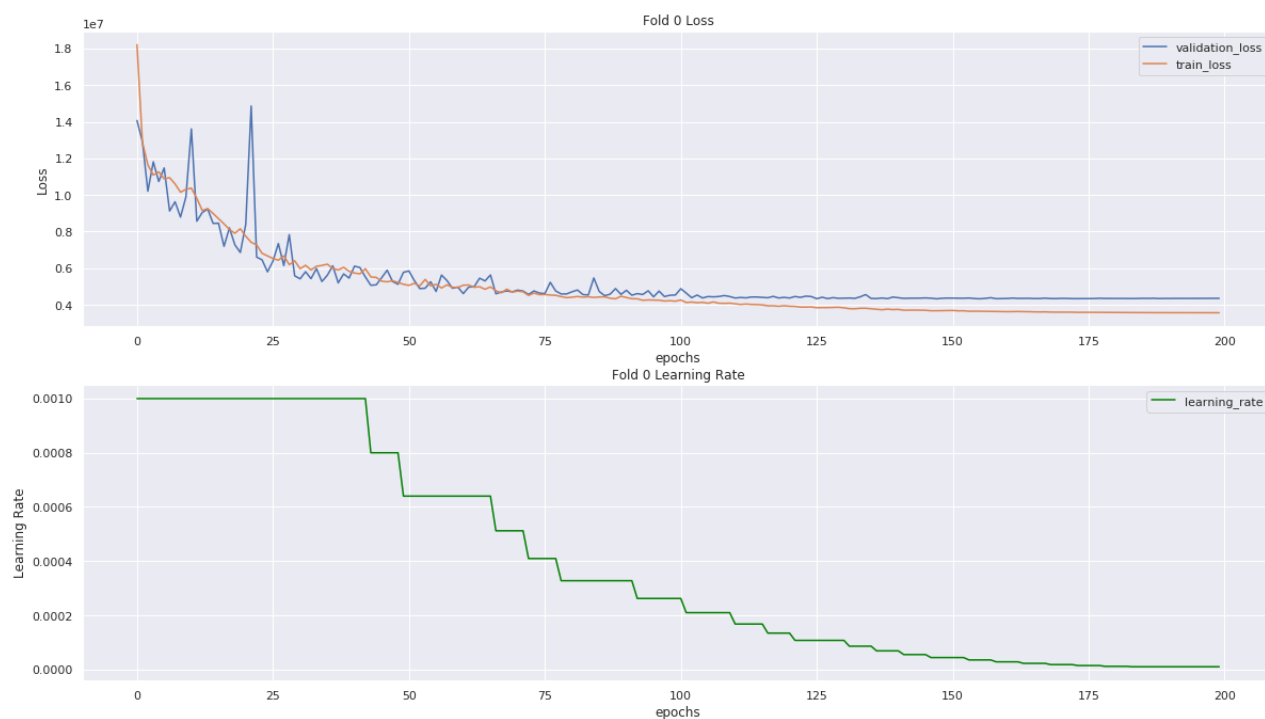
حال نمودار های خطای این ۵ مدل را بررسی میکنیم.



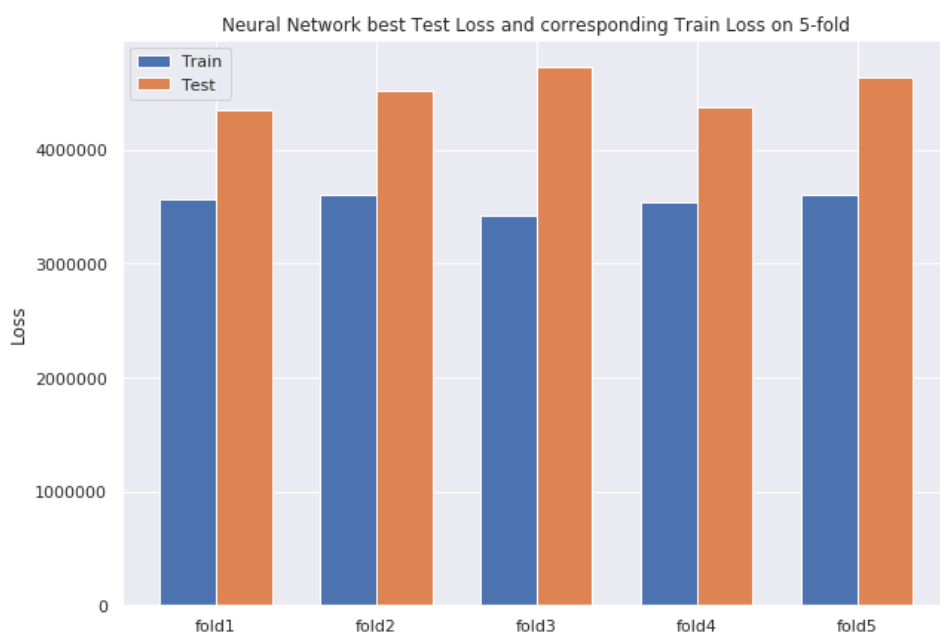


همانطور که در این نمودار ها نیز مشاهده می کنید، بیشتر توزیع فراوانی خطا ها ، در حدود صفر تا یک میلیون است و به ندرت خطا های بسیار زیاد دیده شده اند و میانگین خطاها به طور کلی روی **train** و **test** مطابق نتایجی که در بالا دیدیم ، به ترتیب ، حدود ۳.۵ و ۴.۵ میلیون تومان بوده است . همانطور که انتظار داشتیم ،مدل شبکه عصبی بهتر از مدل رگرسیون خطی عمل کرده است و میانگین خطای آن ،بسیار بهتر شده است .

نمودار های loss - epoch و learning\_rate - epoch برای یکی از این ۵ مدل را در تصاویر زیر میبینید.



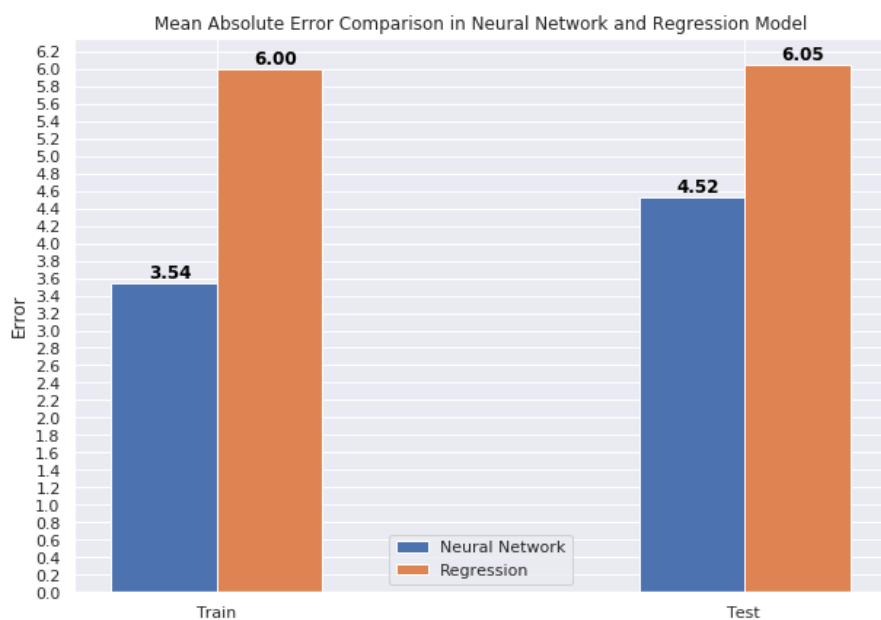
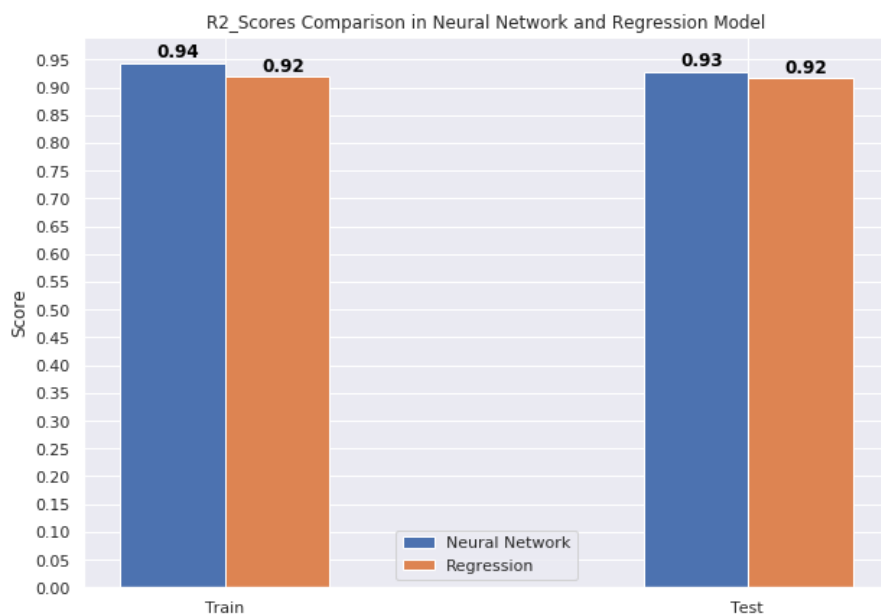
نمودار زیر مقادیر کمترین خطا روی داده های آموزشی و داده های اعتبار سنجی را به ازای ۵ مدل شبکه عصبی مختلف نشان میدهد که تقریباً عملکرد یکسانی دارد اما مدل چهارم نسبت به بقیه عملکرد نسبتاً بهتری دارد پس وزن های این مدل را برای استفاده در رابط کاربری استفاده میکنیم.





## مقایسه و نتیجه گیری:

با توجه به نتایج بدست آمده در بخش های مختلف و سایر توضیحات ، با توجه به اینکه دیتای بررسی شده در اینجا ، کاملاً خطی نبوده و داده ی پیچیده ای محسوب می شود ، مطابق انتظار مدل شبکه عصبی نسبت به رگرسیون خطی عملکرد بهتری داشت و نتایج قابل قبولی برای آن بدست آمد . این تحقیق را با مشاهده نمودار های مقایسه ی مقدار  $r^2\_score$  و  $mean\_absolute\_error$  در دو بخش داده  $train$  و  $test$ ، به پایان می بریم.



همان طور که میبینید توانستیم مقدار خطا را به شکل قابل توجهی کاهش دهیم و ضریب  $R^2$  را نیز ۱ الی ۲ درصد افزایش دهیم که پیشرفت خوبی محسوب میشود.

## رابط کاربری :

برای ایجاد رابط کاربری در این پروژه از کتابخانه Tkinter استفاده میکنیم. کتابخانه Tkinter از کتابخانه هایی است که با نصب کردن زبان python روی سیستم شما نصب میشود و برای استفاده فقط کافی است که آن را `import` کنید.

شمایل ظاهری رابط کاربری طراحی شده به شکل زیر میباشد.



The screenshot shows a window titled "Car Price Estimation AI". Inside the window, there is a large red button labeled "محاسبه کن" (Calculate). To the right of the button, there are several input fields with labels in Persian: "مدل" (Model), "رنگ" (Color), "نوع دنده" (Gear Type), "سال تولید" (Production Year), "وضعیت بدنه" (Body Condition), "شهر" (City), and "کارکرد" (Mileage). Below the button, there are labels for "موارد مشابه:" (Similar items:), "رگرسیون:" (Regression:), and "شبکه عصبی:" (Neural Network:). The "کارکرد" field has a value of "0" entered.

از این رابط کاربری میتوان برای محاسبه قیمت خودرو کار کرده با مشخصات دلخواه استفاده کرد. برای این کار نیاز است که اطلاعات مربوط به خودرو مورد نظر خود را از گزینه های موجود انتخاب کنید. در ابتدا بین گزینه های موجود فقط گزینه مدل در دسترس شما میباشد و با انتخاب کردن مدل مورد نظر گزینه های موجود برای رنگ در اختیار شما قرار میگیرد. به این ترتیب شما با انتخاب کردن مشخصات ماشین مورد نظر خود و وارد کردن مقدار کارکرد این ماشین به کیلومتر و سپس فشردن دکمه "محاسبه کن" نتایجی را در بخش های زیرین مشاهده خواهید کرد. توجه داشته باشید گزینه هایی که برای انتخاب کردن در هر مرحله فعال میباشد تنها شامل گزینه هایی است که داده هایی با آن مشخصات در داده های استخراج شده از سایت دیوار وجود داشته است. برای مثال اگر برای ماشینی در سایت دیوار فقط مدل دنده ای وجود داشته است، در این رابط کاربری نیز گزینه اتوماتیک برای بخش دنده مخصوص

این مدل ماشین غیر فعال میباشد، یک مزیت دیگر که این کار دارد این است که با پر کردن فرم به این شکل پس از انتخاب گزینه "محاسبه کن" حتما نمونه های مشابهی برای آن نمایش داده میشود که به کمک این نمونه های مشابه میتوان دقت عملکرد مدل های رگرسیون و شبکه عصبی را با یکدیگر مقایسه کرد.

نمونه ای از آن به شکل زیر است.

usage	price
0 67000	94,000,000
1 42000	95,000,000
2 42000	95,000,000
3 49000	96,500,000
4 80000	93,000,000

موارد مشابه: 86,600,000 تومان

رگرسیون: 92,800,000 تومان

شبکه عصبی: 50000

خروجی های تولید شده توسط این برنامه سه بخش است.

۱ موارد مشابه :

خروجی این بخش شامل اطلاعات قیمت و کارکرد ماشین هایی میباشد که از سایت دیوار استخراج شده است که این ماشین ها مشخصاتی همانند ماشین مورد نظر ما داشتند و تنها فرقشان کیلومتر کارکردشان میباشد.

این بخش نشان دهنده ی این است که در داده های جمع آوری شده از سایت دیوار record هایی وجود داشته که اطلاعات ماشین در آن ها به شکل زیر بوده است.

پژو 206 تیپ 5

سفید

دنده ای

1396

بدون رنگ

تهران

و قیمت گذاری های زیر بر حسب کیلومتر کارکرد های این چینی برای آن ها در نظر گرفته شده است.

	usage	price
0	67000	94,000,000
1	42000	95,000,000
2	42000	95,000,000
3	49000	96,500,000
4	80000	93,000,000

موارد مشابه :

لازم به ذکر است که در صورتی که تعداد موارد مشابه با ماشین مورد نظر ما در داده های جمع آوری شده کمتر یا مساوی ۵ باشد تمام موارد مشابه در این بخش نشان داده میشود. اما در صورت بیشتر بودن این تعداد فقط ۵ تا از این موارد به صورت Random انتخاب میشود و نمایش داده میشود. یعنی ممکن است با دو بار اجرا کردن این برنامه به ازای یک ماشین خاص، موارد مشابهی که برای آن نمایش داده میشود تغییر کند.

۲ رگرسیون :

در این بخش ما برای پیشبینی قیمت از مدل رگرسیون استفاده میکنیم. برای این کار ما ابتدا نیاز داریم که ورودی های انتخاب شده توسط فرد را به صورتی که داده های reg\_encoded را ایجاد کردیم encode کنیم و یعنی داده های کیفی را به dummy\_variables تبدیل کنیم. پس از این کار داده های encode شده ما آماده predict شدن توسط مدل رگرسیون میباشد. شیوه پیشبینی مدل رگرسیون به این شکل است که پس از Train شدن مدل ضرایبی به ازای هر کدام از ورودی ها به دست می آید، که با ضرب کردن این مقادیر به صورت elementwise در مقادیر ورودی و اضافه کردن مقدار عرض از مبدا به آن پیشبینی مدل برای آن ورودی ها به دست می آید. فرمول آن به شکل زیر است.

$$Y_i = \beta_1 X_1 + \dots + \beta_k X_k + \varepsilon_i$$

که در آن  $\beta_1$  تا  $\beta_k$  ضرایب محاسبه شده توسط مدل رگرسیون و  $X_1$  تا  $X_k$  ورودی های encode شده مدل و  $\varepsilon_i$  مقدار عرض از مبدا مدل ما میباشد.

چون ما در برای Evaluate کردن مدل هایمان از روش K-fold استفاده کردیم و دقت ها و ضرایب و عرض از مبدا های مختلفی را به ازای هر کدام از این ۵ حالت به دست آوردیم و در این جا فقط میتوانیم از یکی از این ۵ تا استفاده کنیم. مدلی را استفاده میکنیم که روی داده هایی که روی آن Train نشده بود بهترین خروجی را داشته که fold

دوم از این لحاظ بهترین عملکرد روی داده هایی که ندیده را داشته است پس ما نیز از ضرایب و عرض از مبدا این مدل برای محاسبه خروجی استفاده میکنیم.

و مقدار پیش بینی شده را در بخش دوم به شکل زیر نمایش میدهیم.

رگرسیون : 86,600,000 تومان

۳ شبکه عصبی :

در این بخش ما برای پیش بینی قیمت از مدل بکه عصبی استفاده میکنیم. برای این کار ما ابتدا نیاز داریم که ورودی های انتخاب شده توسط فرد را به صورتی که داده های neu\_encoded را ایجاد کردیم encode کنیم و یعنی داده های کیفی را Label Encode کنیم تا پس از این کار داده های ما آماده predict شدن توسط مدل شبکه عصبی شوند. همانند مدل های رگرسیون در این جا نیز ما در نهایت به ۵ مدل مختلف رسیدیم که همه ی آن ها از ساختار شبکه ی یکسان و ویژگی های یکسانی برخوردار بودند و تنها تفاوتشان در بخشی از داده است که مدل ها روی آن Train شده اند. در این جا نیز ما برای تخمین قیمت توسط رابط کاربری از کدلی استفاده میکنیم که بیشترین دقت و کمترین خطا را روی داده های مشاهده نشده داشته باشد. ویژگی های شبکه مورد استفاده ما در این بخش های قبلی به طور کامل توضیح داده شده است. تنها کاری که نیاز است در این جا انجام دهیم این است که ابتدا یک شبکه بسازیم که ساختار کلی آن شبیه شبکه اصلیمان باشد و سپس وزن های بدست آمده برای مدل اصلیمان را با دستور

```
network.load_weights(r'Gui\NN_weights\weights4.h5')
```

وارد مدل جدیدمان کنیم. به این شکل تمام ویژگی هایی که آن مدل داشت را در این جا نیز خواهیم داشت و میتوانیم از آن برای پیشبینی قیمت استفاده کنیم. برای این کار کفایست دستور

```
network.predict(neu_encoded_input)
```

را اجرا کنیم که در آن neu\_encoded\_input همان مقادیر ورودی به رابط کاربری میباشد که برای پیشبینی شدن توسط شبکه عصبی Encode شده است.

مقدار پیش بینی شده توسط شبکه عصبی به شکل زیر در رابط کاربری نمایش داده خواهد شد.

شبکه عصبی : 92,800,000 تومان

در زیر یک نمونه دیگر استفاده از این رابط را میبینیم.

usage	price
0	135000 72,000,000
1	125000 73,000,000
2	139 73,000,000
3	139 73,500,000
4	120000 76,000,000

موارد مشابه :

70,000,000 تومان رگرسیون :

75,400,000 تومان شبکه تصببی :

پژو پارس ساده

خاکستری

دنده‌ای

1393

بدون رنگ

تهران

کارکرد: 0

برای اجرای این رابط کاربری نیاز است که کتابخانه های Keras , Tensorflow, nump, pandas, sklearn روی سیستم نصب باشند. پس از برآورده کردن این نیاز ها با اجرای فایل Gui.py رابط کاربری باز میشود و میتوانید از آن استفاده کنید.

## چالش ها:

یکی از چالش های مهم در بحث تخمین قیمت خودرو نوسانات لحظه‌ای و سریع این محصول در بازار می باشد که نیاز است تا به طور مداوم داده جمع آوری شود و سعی شود که این داده ها به صورت آنلاین آموزش داده شوند تا در هر لحظه بتوان قیمت دقیقی بر اساس قیمت های موجود در بازار عرضه کرد.

چالش مهم دیگری که در این تحقیق با آن روبرو بودیم ، این بود که همانطور که گفته شد، دیتاست خود را از آگهی های موجود در سایت دیوار crawl کرده بودیم و این دیتا چند مشکل اساسی داشت:

-خیلی از مردم قیمت هایی که روی خودرو های خود در این سایت قرار داده اند ، را سلیقه ای و بدون نظر کارشناس ، تعیین کرده اند . پس این قیمت ها ، نمی تواند چندان اطلاعات قابل اعتمادی باشد.

-خیلی از آگهی ها در این سایت ممکن است وجود خارجی نداشته باشند و حتی ربات آن ها را ایجاد کرده باشد تا اطلاعات سالم در اختیار همه قرار نگیرد .مثلا تعداد زیادی قیمت دیده شد که همگی ۳۶۰ قرار داده شده بودند.

-همانطور که در بخش های preprocessing و تمیز کردن دیتا دیدیم ، تعداد بسیاری دیتا های Nan و یا قیمت های فاقد اعتبار و غیره وجود داشت که باید آن ها تمیز می کردیم و به طور کلی فرآیند آماده کردن دیتای خام تا مرحله ای که قابل استفاده برای مدل هایمان باشد ، فرآیند طولانی و نسبتاً سختی بود.

یک چالش مهم دیگر هم این بود که دیتاست ، دیتاست بزرگی محسوب می شد و ما امکانات سخت افزاری لازم برای آموزش دادن مدل های پیچیده را نداشتیم و به علت محدودیت سخت افزاری و سرعت اجرای دستور ها ، به ناچار از kernel های موجود در سایت kaggle استفاده کردیم.

پایان