基于因果推断的车辆变道预测与分析

本文档使用的数据来源于 https://github.com/donnydcy/LC_NGSIM

摘要

使用了公开的数据集做的因果推断，简单的数据分析发现换道持续时间与其他变量的相关系数很低。但通过因果推断，构造了因果图，使用的效应估计方法是**backdoor.linear_regression**，使用的反驳方法是**placebo_treatment_refuter**，我们发现，**v**自身车辆的速度，**f_v**当前车道前方车辆的速度，**r_v**当前车道后方车辆的速度，**ft_v**目标车道前方车辆的速度，**rt_v**目标车道后方车辆的速度与变道持续时间有显著的因果效应。同样，车长这个变量也存在较弱的因果性，**x_ft**当前车辆与目标车道前方车辆的横坐标之间的距离与换道持续时间之间存在因果性。

第二步是查看跨越车道的换道对其他车辆行为的影响，也就是**x_ft**对其他变量的影响。根据因果决策树我们发现当速度在**[7.622, 9.222]**之间时，对于当前坐标与目标车道前车横坐标距离的反应最强烈。

本文档为示例文档，还有很多可以挖掘的地方

0.环境的搭建

In [ ]:

```
!apt-get install python3.8
!pip install dowhy
!pip install econml
```

In [ ]:

```
!apt install libgraphviz-dev
!pip install pygraphviz
```

In [3]:

```
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LassoCV
from sklearn.ensemble import GradientBoostingRegressor
import matplotlib.pyplot as plt
import numpy as np
```

In [4]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1.数据的读取与简单的相关性分析

读取数据并展示

In [5]:

```
lane_data = pd.read_csv('/content/drive/MyDrive/data_output2.csv')  # 读取训练数据
lane_data.head()
```

Out[5]:

	len	v	a	v_after	a_after	f_v	r_v	ft_v	rt_v	st_v	f_a	r_a
0	4.08432	4.584192	-0.515112	6.071616	0.435864	4.267200	3.313176	6.565392	5.961888	6.400800	0.064008	0.000000
1	4.08432	6.102096	0.411480	10.674096	-0.082296	6.096000	6.705600	7.610856	6.086856	6.998208	0.000000	0.000000
2	4.69392	4.422648	-0.048768	7.836408	0.000000	4.660392	2.523744	6.473952	6.824472	10000.000000	1.764792	1.066800
3	4.23672	5.830824	0.112776	7.391400	-0.716280	5.900928	6.096000	7.565136	3.892296	7.251192	0.838200	0.000000
4	3.93192	8.135112	0.000000	8.558784	1.444752	6.419088	5.178552	4.828032	5.105400	10000.000000	-0.051816	0.051816

变量解释

- **len** 车长
- **v** 当前车速
- **a** 加速度
- **v_after** 换道后的速度
- **a_after** 换到后的加速度
- **f_v** 前方车辆的速度
- **r_v** 后方车辆的速度
- **ft_v** 目标车道上前方车辆的速度
- **rt_v** 目标车道上后方车辆的速度
- **st_v** 目标车道上重叠车辆的速度
- **f_a** 前方车辆的加速度
- **r_a** 后方车辆的加速度
- **ft_a** 目标车道上前方车辆的加速度
- **rt_a** 目标车道上后方车辆的加速度
- **st_a** 目标车道上重叠车辆的加速度
- **f_len** 前方车辆的车长
- **r_len** 后方车辆的车长
- **ft_len** 目标车道前方车辆的车长
- **rt_len** 目标车道后方车辆的车长
- **st_len** 目标车道上重叠车辆的长度
- **x_ft** 换道前当前**x**坐标与目标车道前方车辆**x**坐标之差
- **y_ft** 换道前当前**y**坐标与目标车道前方车辆**y**坐标之差
- **x_rt** 换道前当前**x**坐标与目标车道后方车辆**x**坐标之差
- **y_rt** 换道前当前**y**坐标与目标车道后方车辆**y**坐标之差
- **duration** 换道持续时间

首先检查相关性，用的是spearman相关系数

In [ ]:

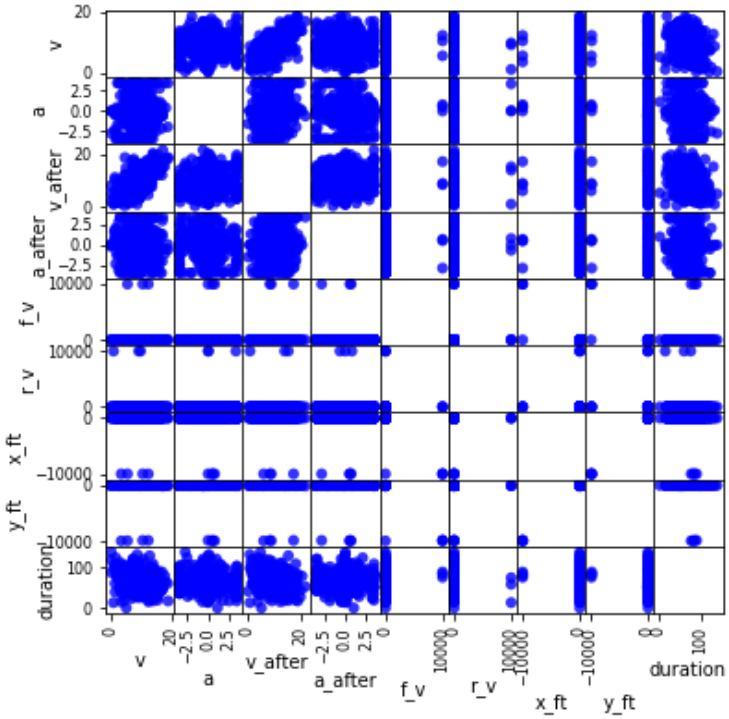
```
names=lane_data.columns
```

```
correlations = lane_data.corr(method='spearman')
correction=correlations

fig = plt.figure()
ax = plt.subplots(figsize=(20, 15))
ax = sns.heatmap(correction,cmap=plt.cm.Greys, linewidths=0.1,vmax=1, vmin=0 ,annot=True
,annot_kws={'size':7,'weight':'bold'})
#热力图参数设置（相关系数矩阵，颜色，每个值间隔等）
#ticks = numpy.arange(0,16,1) #生成0-16，步长为1
plt.xticks(np.arange(26)+0.5,names) #横坐标标注点
plt.yticks(np.arange(26)+0.5,names) #纵坐标标注点
#ax.set_xticks(ticks) #生成刻度
#ax.set_yticks(ticks)
#ax.set_xticklabels(names) #生成x轴标签
#ax.set_yticklabels(names)
ax.set_title('Characteristic correlation')#标题设置
plt.savefig('cluster.tif',dpi=300)
plt.show()
```

In [ ]:

```
fig = pd.plotting.scatter_matrix(lane_data.iloc[:,[1,2,3,4,5,6,20,21,24]],figsize=(6,6),
c ='blue',marker = 'o',diagonal='',alpha = 0.8,range_padding=0.2)
plt.show()
```



可以看到，相关性很差，变道时间与所有变量的spearman相关系数都很小

2.因果推断----哪些变量与换道持续时间之间有因果性

In [6]:

```
from dowhy import CausalModel
import dowhy.datasets
import numpy as np
import pandas as pd
import warnings
from sklearn.exceptions import DataConversionWarning
from IPython.display import Image, display

warnings.filterwarnings(action='ignore', category=DataConversionWarning)
warnings.filterwarnings(action='ignore', category=FutureWarning)

# Config dict to set the logging level
import logging
```

```
import logging.config

DEFAULT_LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'loggers': {
        '': {
            'level': 'WARN',
        },
    }
}

logging.config.dictConfig(DEFAULT_LOGGING)
logging.info("Getting started with DoWhy. Running notebook...")
```

dag { bb="0,0,1,1" a [pos="0.274,0.255"] a_after [pos="0.243,0.125"] duration [outcome,pos="0.459,0.434"] f_a [pos="0.903,0.433"] f_len [pos="0.786,0.448"] f_v [pos="0.930,0.318"] ft_a [pos="0.291,0.843"] ft_len [pos="0.623,0.685"] ft_v [pos="0.310,0.741"] r_a [pos="0.136,0.728"] r_len [pos="0.771,0.573"] r_v [pos="0.067,0.592"] rt_a [pos="0.581,0.843"] rt_len [pos="0.693,0.615"] rt_v [pos="0.490,0.705"] st_len [pos="0.777,0.248"] v [pos="0.078,0.340"] v_after [pos="0.146,0.203"] x_ft [pos="0.383,0.199"] x_rt [pos="0.603,0.168"] y_ft [pos="0.496,0.162"] y_rt [pos="0.684,0.175"] a -> duration a -> v a -> v_after a_after -> v_after f_a -> f_v f_len -> duration f_v -> duration f_v -> v ft_a -> ft_v ft_len -> duration ft_v -> duration ft_v -> rt_v ft_v -> v r_a -> r_v r_len -> duration r_v -> duration r_v -> v rt_a -> rt_v rt_len -> duration rt_v -> duration rt_v -> v st_len -> duration v -> duration v -> v_after x_ft -> duration x_rt -> duration y_ft -> duration y_rt -> duration }

上面的代码是因果图的绘制代码，下面的是**dowhy**适用的因果图

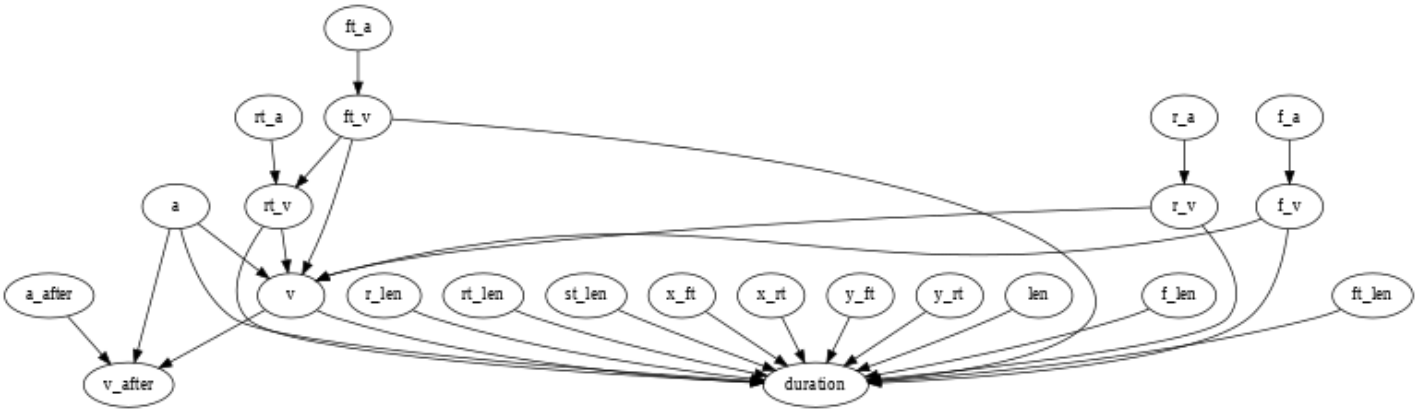
In [7]:

```
dot_graph = 'digraph { a -> duration;a -> v;a -> v_after;a_after -> v_after;f_a -> f_v;f
_len -> duration;f_v -> duration;'\
            'f_v -> v;ft_a -> ft_v;ft_len -> duration;ft_v -> duration;ft_v -> rt_v;ft_v
-> v;r_a -> r_v;r_len -> duration;'\
            'r_v -> duration;r_v -> v;rt_a -> rt_v;rt_len -> duration;rt_v -> duration;r
t_v -> v;st_len -> duration;'\
            'v -> duration;v -> v_after;x_ft -> duration;x_rt -> duration;y_ft -> durati
on;y_rt -> duration;len -> duration}'
```

In [8]:

```
model=CausalModel(
        data = lane_data,
        treatment='v',
        outcome='duration',
        graph=dot_graph
        )
model.view_model()


display(Image(filename="causal_model.png"))
```



上图就是预先假设的因果图

下面进行因果效应的计算与反驳，反驳方法使用的是placebo_treatment_refuter

下面进行因果效应的计算与反驳，反驳方法使用的是placebo_treatment_refuter

In [9]:

```
treatment_text=['len','v','a','f_v','r_v','ft_v','rt_v','f_a','r_a','ft_a','rt_a','f_len
','r_len','ft_len','rt_len','st_len','x_ft','y_ft','x_rt','y_rt']
where_are_nan = np.isnan(lane_data)
where_are_inf = np.isinf(lane_data)
lane_data[where_are_nan] = 0
new_eff=np.zeros(np.size(treatment_text))
estimated_eff=np.zeros(np.size(treatment_text))

#total_eff=np.zeros(n*np.size(treatment_text))
#total_eff=total_eff.reshape(n,np.size(treatment_text))
```

In [ ]:

```
#for j in range(n):
for i in range(np.size(treatment_text)):

  model=CausalModel(
        data = lane_data,
        treatment=treatment_text[i],
        outcome='duration',
        graph=dot_graph
        )

  #model.view_model()

  identified_estimand = model.identify_effect(proceed_when_unidentifiable=True)
  causal_estimate = model.estimate_effect(identified_estimand, method_name="backdoor.line
ar_regression", test_significance=True)
  refutation = model.refute_estimate(identified_estimand, causal_estimate, method_name="
placebo_treatment_refuter",
                                     placebo_type="permute", num_simulations=20)
  new_eff[i]=refutation.new_effect
  estimated_eff[i]=refutation.estimated_effect
  #total_eff[j]=new_eff-estimated_eff
```

In [ ]:

```
new_eff
```

Out[ ]:

```
array([-1.79223341e-01,  3.09267368e-01,  6.40897559e-01, -3.52687508e-02,
       -7.51329237e-02,  2.31701625e-03, -7.88865778e-02, -4.55854834e+00,
       -8.08192973e-01, -1.80562553e-01, -5.37392557e-01, -4.77646728e-02,
        3.90032210e-03,  8.71708074e-02, -4.58863721e-02,  3.03750359e-04,
       -2.86790855e-02,  9.59586543e-03,  1.14834718e-03, -5.08341314e-04])
```

每次运行大概38分钟

下面是运行十次后的结果，画出对应的箱形图

In [10]:

```
plac_try1_est=np.array([-6.85907141e-01, -6.01592106e-01, -1.42772482e+00, -4.09914006e-
01,
       -1.07469737e+00, -7.86928153e-01, -8.56394536e-01, -2.38859419e+00,
        2.61667766e-01,  1.14649600e-01, -4.39508941e+00,  2.07138769e-01,
       -7.46703309e-01,  7.34067575e-01, -5.02354080e-01, -2.89950018e-04,
       -5.07828828e-01,  8.17228064e-02, -3.70851945e-02,  1.15069860e-02])
plac_try1_new=np.array([[ 3.72977698e-01,  4.41043122e-01, -1.62318758e+00,  4.57060383e
-02,
        1.69780893e-02, -1.42629066e-02, -4.30271365e-02,  3.38057699e+00,
       -1.59155706e-01,  5.22745871e-01,  1.39249230e+00, -1.01637051e-01,
        1.38673656e-01, -2.35434625e-02, -5.96578619e-02, -3.71784543e-04,
       -3.02176460e-02, -8.38957224e-03,  5.06232636e-02,  8.54283241e-03]])
plac_try2_new=np.array([[-5.83606390e-02,  4.31357908e-01,  1.41579493e+00, -1.06347817e
-01,
```

```
       -2.88567137e-02, -8.87612762e-03,  1.48278016e-02,  2.29942547e+00,
        4.75020561e+00, -8.85152789e-01, -2.81261906e+00, -3.27187658e-02,
       -2.21172716e-01, -3.07000339e-02, -1.13829213e-01,  2.43472466e-04,
       -1.25354207e-01,  6.32851607e-04,  1.17520348e-01, -1.09226947e-04]])

plac_try3_new=np.array([[ 4.34575273e-02,  3.64910435e-01, -2.19157401e-03, -1.72147406e
-02,
       -1.55856909e-02,  6.92652310e-02, -1.48760409e-01, -6.71541285e-01,
       -6.48592193e+00, -4.52558050e-02, -3.24580899e-01,  4.31779411e-02,
       -6.93255048e-02,  1.13301511e-01,  1.90276596e-02, -3.75722362e-04,
       -7.90096656e-02,  9.94645025e-03,  4.17907783e-03,  3.15864548e-03]])

plac_try4_new=np.array([[ 2.18955349e-01,  3.58923585e-01, -1.70086936e+00,  2.01382574e
-02,
        8.36439074e-03, -4.74545910e-02, -2.13586312e-02,  1.54966171e+00,
       -1.60221210e+00,  1.13126737e+00, -2.28193871e+00, -1.02051937e-01,
        3.61555888e-02,  1.36174232e-02,  5.47347000e-03,  4.60633366e-04,
       -5.13651295e-02,  3.26275950e-03,  1.19901495e-01,  2.49835881e-02]])

plac_try5_new=np.array([[-3.44639304e-01,  3.54359223e-01,  3.21814920e-01, -4.39626746e
-02,
       -2.79894640e-02, -7.30348508e-04, -2.92868155e-02, -3.13489019e+00,
       -1.75040062e-01, -4.99899393e-01, -7.57026595e-01, -1.89257167e-02,
       -9.22394841e-02, -6.71266677e-02, -2.37158618e-02,  6.02335771e-04,
        3.04755354e-02, -1.12645294e-02, -6.02052633e-02, -2.13293578e-02]])

plac_try6_new=np.array([[ 8.59129387e-02,  2.57345237e-01,  1.85584400e+00, -3.90807094e
-02,
        1.74070495e-02, -8.15750395e-02, -9.49157507e-03,  3.65216172e-01,
       -1.91893921e-01,  4.19236561e+00,  2.87958157e-01, -2.05121478e-02,
       -6.60003575e-02, -2.48381996e-02, -3.01171091e-02, -2.85265034e-04,
        6.13801778e-02, -1.51660416e-03,  7.59267231e-02,  2.55352639e-02]])

plac_try7_new=np.array([[-4.12676883e-01,  2.13417578e-01,  1.61845927e-01,  4.16170958e
-02,
       -1.28391915e-02,  5.02097103e-02, -8.68294178e-02, -6.36936189e-01,
       -4.14251415e-01,  2.72285209e-01, -4.32415268e-01, -2.54483600e-02,
       -1.21867299e-01,  4.47050114e-02, -6.51933675e-02,  5.24264370e-04,
       -1.99010513e-02, -1.68513302e-02,  1.27496818e-01, -9.69989617e-03]])

plac_try8_new=np.array([[ 2.43683590e-01,  2.72374197e-01,  5.35848069e-01, -5.38686563e
-02,
        4.44368743e-02, -2.66401138e-02, -2.80859170e-02,  1.35215145e+00,
       -4.13468156e-01,  1.18717815e+00,  3.13494660e+00, -7.52080700e-02,
        3.69713045e-02, -1.39400449e-01, -3.34294183e-02, -1.98757544e-04,
       -1.00332543e-01,  1.01907297e-02,  1.96876954e-02, -2.61000529e-02]])

plac_try9_new=np.array([[-2.47179684e-02,  2.55393980e-01, -3.55888375e+00, -3.13398985e
-02,
       -4.42066715e-02, -1.68179733e-02, -4.60960700e-03,  3.39397011e+00,
       -5.13507252e-01, -1.63465238e-01, -8.23412077e-01, -4.60291248e-02,
        1.55450544e-01, -9.12690642e-02, -1.78813192e-02, -3.20959254e-04,
       -2.06120513e-02,  5.61482968e-04, -3.81701016e-03, -2.62382966e-02]])

plac_try10_new=np.array([[-1.02644880e-01,  2.93548981e-01, -1.28897892e+00, -6.53825103
e-02,
       -5.57732002e-02,  5.14594876e-02,  4.12387896e-02, -7.49128589e-01,
       -2.84083672e+00,  7.17431139e-01,  1.59401666e-01,  2.83054827e-02,
        1.08583614e-01, -4.93748328e-02, -2.16026758e-02, -1.92499214e-04,
       -1.33809043e-01, -9.65342032e-03, -8.10144283e-03, -1.60377493e-02]])

plac_try11_new=np.array([[ 1.68584087e-01,  1.82566087e-01,  3.08234034e+00,  4.82190921
e-03,
        3.54236368e-02, -5.93515430e-02,  6.45175520e-02,  1.27884921e-01,
       -1.96150137e-01, -1.78359565e+00, -2.05910659e+00, -8.35748109e-02,
       -3.43765064e-03,  7.09627116e-02, -1.72580458e-02,  3.93093017e-04,
        3.43788568e-02, -4.67562490e-03,  3.61027271e-02, -1.93965340e-02]])
plac_try12_new=np.array([[-3.65093724e-01,  3.59860918e-01,  1.53446065e+00,  2.72830049
e-02,
       -4.99240470e-02,  2.54786898e-02, -7.87914829e-02,  4.18709413e+00,
        3.92145869e-03,  2.13425717e+00, -8.71019631e-03, -4.53185372e-02,
       -5.07770865e-02, -5.67405366e-02, -3.90676534e-02,  3.45500794e-04,
```

```
        1.00349672e-01,  1.36010520e-02, -7.48176358e-03, -1.86417270e-02]])
plac_try13_new=np.array([[ 6.88837156e-01,  3.22100540e-01,  1.57556555e+00, -9.08067699
e-02,
        1.10971983e-02,  5.96618376e-02, -9.55343790e-02,  1.38266475e-01,
       -7.58549534e-01, -1.55653292e-01, -2.06638980e+00, -1.14502048e-01,
       -1.65523019e-01,  5.41723333e-03,  2.54177224e-02,  3.48746811e-04,
       -2.35584011e-02, -4.90641572e-03, -5.39535581e-02,  1.11718698e-02]])
plac_try14_new=np.array([[ 4.26721057e-01,  4.00261470e-01,  2.00256360e+00, -1.69612232
e-02,
        6.27172415e-04, -1.16169255e-02, -6.92722528e-02, -1.55645038e+00,
       -1.63276241e+00,  3.85406756e-01,  4.68978152e-01, -2.09453009e-02,
        3.85203653e-02, -3.74940047e-02, -6.39266484e-02,  1.95303789e-04,
       -1.42821273e-03,  2.47309033e-02,  2.38649619e-02, -7.86858462e-03]])
plac_try15_new=np.array([[ 5.58800419e-02,  1.69562123e-01,  1.78863886e+00, -9.02129918
e-02,
        1.99249591e-02,  3.12373915e-02, -3.95576118e-02, -1.14996109e+00,
        2.90590570e-01, -8.71554078e-01,  1.03125713e+00,  1.96599284e-02,
       -1.68481369e-01, -1.05682508e-01,  8.16009248e-02, -2.24668733e-04,
        1.01232618e-02,  1.87126988e-02, -7.66367944e-02, -1.55967061e-02]])
plac_try16_new=np.array([[-4.67362806e-01,  3.28936804e-01,  5.45242002e-01, -3.91050292
e-02,
        1.42267998e-02, -5.38206923e-02,  6.07541579e-02, -4.31828902e-01,
        1.41723072e+00, -2.76054364e+00, -1.09746570e+00, -1.14599392e-01,
        3.33578868e-02, -2.07016091e-01, -1.55163719e-01, -2.95530511e-04,
       -1.35916646e-02,  9.34049657e-03,  4.90457065e-02,  1.66783296e-02]])
plac_try17_new=np.array([[-3.35919787e-01,  3.69380002e-01,  5.81990467e-01,  5.43204604
e-02,
       -5.33919068e-02,  1.88887346e-02, -5.48035454e-02, -1.97885983e+00,
        2.17923736e+00, -8.33779567e-01, -1.99200990e-01,  6.89786121e-03,
        8.24817648e-02, -1.49845826e-01,  5.25815573e-02, -2.08368001e-04,
       -8.40728131e-03, -1.26559476e-02,  1.14204423e-01,  6.61324990e-03]])
plac_try18_new=np.array([ [2.72777756e-01,  4.30621385e-01,  1.36782117e+00, -6.03863086
e-02,
        4.64195128e-02, -5.93052828e-02, -4.24710309e-02,  1.72696698e-01,
       -5.11989351e-01,  8.35700686e-01,  8.19515562e-01,  1.27373353e-02,
       -7.22394565e-02, -7.83901737e-02, -1.06093414e-02, -1.67768837e-04,
       -1.19897704e-02,  1.25432071e-02,  9.96696011e-04,  2.75250119e-03]])
plac_try19_new=np.array([[-1.79223341e-01,  3.09267368e-01,  6.40897559e-01, -3.52687508
e-02,
       -7.51329237e-02,  2.31701625e-03, -7.88865778e-02, -4.55854834e+00,
       -8.08192973e-01, -1.80562553e-01, -5.37392557e-01, -4.77646728e-02,
        3.90032210e-03,  8.71708074e-02, -4.58863721e-02,  3.03750359e-04,
       -2.86790855e-02,  9.59586543e-03,  1.14834718e-03, -5.08341314e-04]])
plac_try20_new=np.array([[ 4.34575273e-02,  3.64910435e-01, -2.19157401e-03, -1.72147406
e-02,
       -1.55856909e-02,  6.92652310e-02, -1.48760409e-01, -6.71541285e-01,
       -6.48592193e+00, -4.52558050e-02, -3.24580899e-01,  4.31779411e-02,
       -6.93255048e-02,  1.13301511e-01,  1.90276596e-02, -3.75722362e-04,
       -7.90096656e-02,  9.94645025e-03,  4.17907783e-03,  3.15864548e-03]])
```

In [11]:

```
new_data=np.concatenate((plac_try1_new,plac_try2_new,plac_try3_new,plac_try4_new,plac_try
5_new,
                         plac_try6_new,plac_try7_new,plac_try8_new,plac_try9_new,plac_t
ry10_new,
                         plac_try11_new,plac_try12_new,plac_try13_new,plac_try14_new,pl
ac_try15_new,
                         plac_try16_new,plac_try17_new,plac_try18_new,plac_try19_new,pl
ac_try20_new),axis=0)
```

plac_try3_new=np.array([[4.34575273e-02, 3.64910435e-01, -2.19157401e-03, -1.72147406e-02, -1.55856909e-02, 6.92652310e-02, -1.48760409e-01, -6.71541285e-01, -6.48592193e+00, -4.52558050e-02, -3.24580899e-01, 4.31779411e-02, -6.93255048e-02, 1.13301511e-01, 1.90276596e-02, -3.75722362e-04, -7.90096656e-02, 9.94645025e-03, 4.17907783e-03, 3.15864548e-03]])

In [12]:

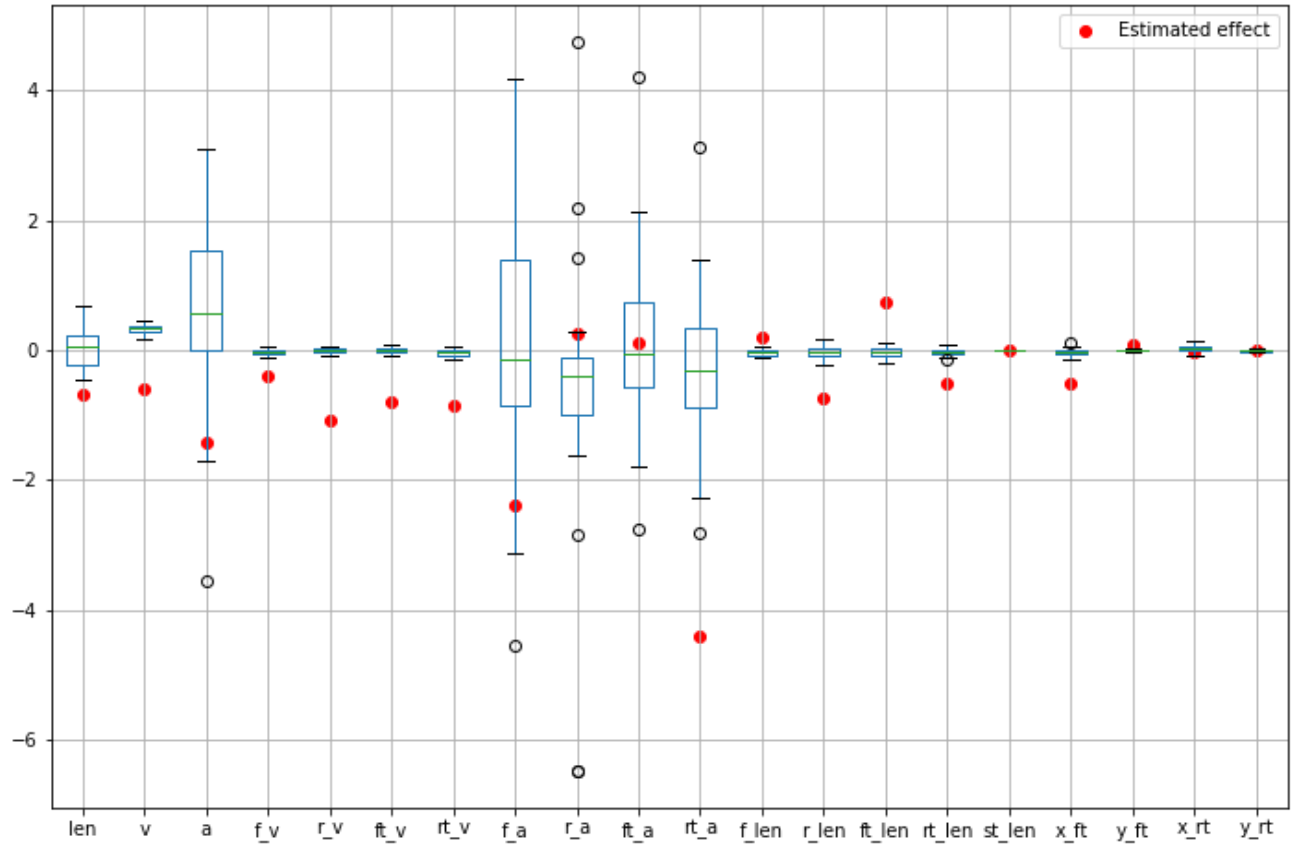
```
new_data=pd.DataFrame(new_data,columns =treatment_text)
#est_data=pd.DataFrame(est_data,columns =treatment_text)
```

In [13]:

```
plt.figure(figsize=(12,8))
new_data.boxplot()
plt.scatter(range(1,21),plac_try1_est,marker='o',c='r',label='Estimated effect')
plt.legend()
#plt.show()
```

Out[13]:

```
<matplotlib.legend.Legend at 0x7eff8f732c90>
```



做因果分析时，对于每个treatment feature，估计的因果效应不在0附近，反驳后的因果效应相较于之前估计的因果效应接近于0，那么我们就可以说这个treatment feature和outcome feature之间有因果关系。

从上面的图我们可以看到的是

1.在准备进行换道这个时刻，自身车辆的车长与加速度没有检测到与换道持续时间之间的因果效应。

2.v自身车辆的速度，**f_v**当前车道前方车辆的速度，**r_v**当前车道后方车辆的速度，**ft_v**目标车道前方车辆的速度，**rt_v**目标车道后方车辆的速度与变道持续时间有显著的因果效应。

3.相关车辆的加速度与环道持续时间之间没有检测到显著的因果效应。

4.r_len当前车道前方与后方车辆的车长，**ft_len**目标车道前方后方车辆的车长与换道持续时间之间存在因果效应，但**effect**比较弱。

5.x_ft当前车辆与目标车道前方车辆的横坐标之间的距离与换道持续时间之间存在因果性。

可以发现，很有趣的是，**f_v**当前车道前方车辆的速度，**r_v**当前车道后方车辆的速度，**ft_v**目标车道前方车辆的速度，**rt_v**目标车道后方车辆的速度这四个**treatment**的因果效应都是负数，也就是说，不管前方还是后方车辆，相关车辆的速度越大，换道时间越短。

实际上这是可以解释的，前方车辆的速度越大，当然留给自己车辆的空间也就越大，当然可以减少换道时间。

而后方车辆速度越大，也就意味着自身车辆的速度也大，因为司机选择换道时，司机认知上的规则是自身的速度肯定是要大于后方车辆的速度的，自身速度越大，换道时间越短。

2.用用分析——跨越车道的换道是怎样影响相关车道车辆行为的

3.因果分析----跨越车道的决定是怎样影响相关车道车辆行为的

In [ ]:

```
from econml.solutions.causal_analysis import CausalAnalysis

ca = CausalAnalysis(
    feature_inds=treatment_text,
    categorical=[],
    heterogeneity_inds=None,
    classification=False,
    nuisance_models="automl",
    heterogeneity_model="forest",
    n_jobs=-1,
    random_state=123,
)
ca.fit(lane_data[treatment_text], lane_data['duration'])
```

In [15]:

```
global_summ = ca.global_causal_effect(alpha=0.05)
global_summ.sort_values(by="p_value")
```

Out[15]:

feature	point	stderr	zstat	p_value	ci_lower	ci_upper
ft_len	0.001115	0.000141	7.925260	2.276696e-15	0.000839	0.001390
x_ft	-0.000690	0.000125	-5.531618	3.172902e-08	-0.000935	-0.000446
ft_v	0.000593	0.000128	4.639033	3.500429e-06	0.000343	0.000844
ft_a	0.000670	0.000191	3.499948	4.653490e-04	0.000295	0.001045
a	-1.362801	0.711692	-1.914875	5.550840e-02	-2.757692	0.032089
v	-1.851099	1.066043	-1.736420	8.248964e-02	-3.940506	0.238308
x_rt	-0.383083	0.316078	-1.211990	2.255161e-01	-1.002585	0.236418
r_v	-0.006707	0.006011	-1.115798	2.645084e-01	-0.018488	0.005074
r_a	-0.006718	0.006112	-1.099056	2.717437e-01	-0.018698	0.005262
r_len	-0.006675	0.006134	-1.088181	2.765150e-01	-0.018697	0.005347
rt_v	-0.301700	0.314689	-0.958724	3.376979e-01	-0.918478	0.315079
f_v	0.001640	0.002089	0.785026	4.324384e-01	-0.002454	0.005733
rt_len	-0.095639	0.186440	-0.512975	6.079689e-01	-0.461054	0.269777
f_a	-0.279393	0.576798	-0.484386	6.281117e-01	-1.409897	0.851111
st_len	-0.000159	0.000334	-0.477931	6.326990e-01	-0.000813	0.000494
y_rt	0.028903	0.068245	0.423519	6.719163e-01	-0.104854	0.162660
rt_a	0.042889	0.245806	0.174484	8.614852e-01	-0.438883	0.524661
f_len	0.023252	0.331038	0.070241	9.440019e-01	-0.625571	0.672076
y_ft	0.000022	0.004084	0.005448	9.956531e-01	-0.007982	0.008026
len	-0.004425	1.720562	-0.002572	9.979478e-01	-3.376664	3.367813

In [16]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(lane_data[treatment_text], lane_data
['duration'], test_size=0.2, random_state=42)
Y_train=y_train.values
T_train=X_train[['x_ft']].values
x_train=X_train[['v']].values
W_train=X_train[[c for c in X_train.columns if c not in ['v', 'x_ft']]].values
x_test=X_test[['v']].values
```

```
#X_test=X_test[['v']]
confounder_names=['len','a','f_v','r_v','ft_v','rt_v','f_a','r_a','ft_a','rt_a','f_len',
'r_len','ft_len','rt_len','st_len','y_ft','x_rt','y_rt']
```

In [17]:

```
from econml.dml import CausalForestDML
est_nonparam = CausalForestDML(model_y=GradientBoostingRegressor(), model_t=GradientBoost
ingRegressor())
# fit through dowhy
est_nonparam_dw = est_nonparam.dowhy.fit(Y_train, T_train, X=x_train, W=W_train)
```

```
WARNING:dowhy.causal_model:Causal Graph not provided. DoWhy will construct a graph based
on data inputs.
WARNING:dowhy.causal_identifier:Max number of iterations 100000 reached. Could not find a
valid backdoor set.
```

In [18]:

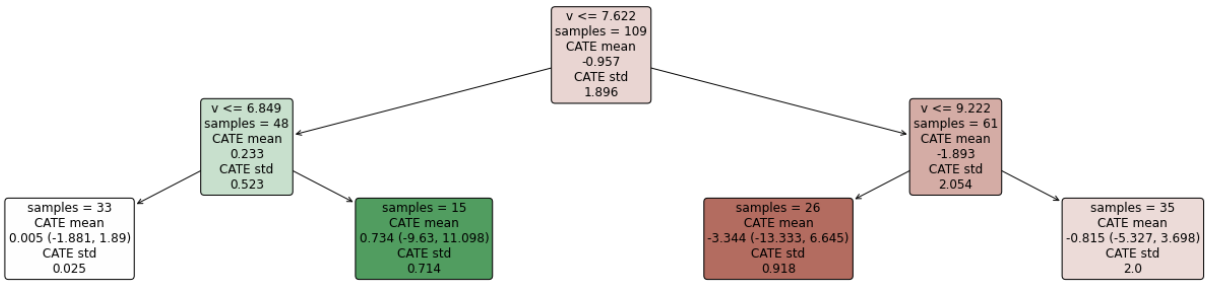
```
te_pred = est_nonparam_dw.effect(x_test).flatten()
te_pred_interval = est_nonparam_dw.effect_interval(x_test)
te_lower, te_upper = est_nonparam_dw.effect_interval(x_test)
#truth_te_estimate = np.apply_along_axis(te_pred, 1, x_test, 1000, "mean")
```

In [ ]:

```
# Compare the estimate and the truth
%matplotlib inline
plt.figure(figsize=(10, 6))
plt.plot(x_test.flatten(), te_pred, label="Sales Elasticity Prediction")
plt.plot(x_test.flatten(), y_test, "--", label="True Elasticity")
plt.fill_between(
    x_test.flatten(),
    te_pred_interval[0].flatten(),
    te_pred_interval[1].flatten(),
    alpha=0.2,
    label="95% Confidence Interval",
)
plt.fill_between(
    x_test.flatten(),
    te_lower,
    te_upper,
    alpha=0.2,
    label="True Elasticity Range",
)
plt.xlabel("Income")
plt.ylabel("Songs Sales Elasticity")
plt.title("Songs Sales Elasticity vs Income")
plt.legend()
#plt.show()
```

In [20]:

```
from econml.dml import LinearDML, CausalForestDML
from econml.cate_interpreter import SingleTreeCateInterpreter, SingleTreePolicyInterprete
r
intrp = SingleTreeCateInterpreter(include_model_uncertainty=True, max_depth=2, min_sample
s_leaf=10)
intrp.interpret(est_nonparam_dw, x_test)
plt.figure(figsize=(25, 5))
intrp.plot(feature_names=["v"],fontsize=12)
```



我把**x_ft**与目标车道前方车辆的横坐标距离当作**treatment feature**，将**v**当作控制的因素，将换道持续时间当作**outcome**结果变量，将其他变量当作**confounder**混杂因素。

通过上图，我们可以发现，当速度在[7.622, 9.222]之间时，对于横坐标距离的反应最强烈，平均treatment effect为-3.344.