```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

$$p_t = \sum_{t'<t}\left[G(t-t')V_{t'}^{\alpha}\epsilon_{t'}\right] + \varepsilon_t. \quad (1)$$

$$R_l = \langle (p_{t+l} - p_t)\epsilon_t \rangle_{over\ t}. \quad (2)$$

$$C(l) \equiv \langle \epsilon_t \epsilon_{t+l} V_{t+l}^{\alpha} \rangle \quad (4)$$

$$C(l) \sim \bar{V}^{\alpha} \langle \epsilon_t \epsilon_{t+l} \rangle, \text{ or } C(l) \sim \bar{V}^{\alpha} c(l) \text{ where } c(l) \equiv \langle \epsilon_t \epsilon_{t+l} \rangle. \quad (5)$$

4. **Question 1 (25 points)**: With (1), (4) and (5), prove that the response function defined in (2) can be written as

$$R_l \sim \bar{V}^{\alpha}\left[\sum_{0<t'\le l} G(t')\, c(t'-l) + \sum_{t'>l} G(t')\, c(t'-l) - \sum_{0<t'} G(t')\, c(t')\right]. \quad (6)$$

Given the trade price eqⁿ (1) and substituting into (2), we get

$$R_l = \left\langle \left(\Sigma_{t'<t+l} \left[G(t+l-t') V_{t'}^{\alpha} \varepsilon_{t'} \right] + \varepsilon_{t+l} - \Sigma_{t'<t} \left[G(t-t') V_{t'}^{\alpha} \varepsilon_{t'} \right] - \varepsilon_t \right) \varepsilon_t \right\rangle$$

We split the sum in the first term at $t+l$ into two parts, $t \to t+l$ & $0 \to t$

Then, we subtract the sum up to t, effectively only considering trades b/w t & $t+l$.

$$R_l = \left\langle \underbrace{\Sigma_{t'<t+l,\, t'\geq t}}_{} \left[G(t+l-t') V_t^{\alpha} \varepsilon_{t'} \right] \varepsilon_t - \underbrace{\Sigma_{t'<t}}_{} \left[G(t-t') V_t^{\alpha} \varepsilon_{t'} \right] \varepsilon_t \right\rangle$$

$t'<t+l$ & $t'\geq t$

$\Rightarrow t'-t<l$ & $t'-t \geq 0$

$\Rightarrow 0 \leq t'-t < l$

These are the trades that occured after time t but befor or at $t+l$. These are the trades that have direct impact on price at $t+l$ but Not at t.

This range includes trades that occur before time $-t$ and hence impact both the price at t & $t+l$.

∴ Applying the Autocorrelation fxⁿ. (4) & (5) where $C(l) \sim \bar{V}^{\alpha} c(l)$

$$R_l = \bar{V}^{\alpha} \left[\Sigma_{0<t'<l} G(t') c(t'-l) + \Sigma_{t'>l} G(t') c(t'-l) - \Sigma_{0<t'} G(t') c(t') \right]$$

Please not, we drop the expectation notation, as the terms inside the brackets are already in a expected form vis-a-vis eqⁿ (5).

---

5. **Question 2 (25 points)**: With the data provided with this assignment (see appendix of this assignment on column definitions in the data file), construct $\widetilde{R}_l$ for $0 \leq l \leq 500$ as defined in equation (3) using all the available trades provided.

In [ ]:
```python
df1 = pd.read_csv('pp1_md_201607_201607.csv')
df1.drop("Unnamed: 0", axis=1, inplace=True)

df2 = pd.read_csv('pp1_md_201608_201608.csv')
df2.drop("Unnamed: 0", axis=1, inplace=True)
```

In [ ]:
```python
df1
```

Out[ ]:

	Date	Time	Size	VWAP	Sign	midQ	BP1	SP1
0	20160701	90100020	48.0	5267.916667	-1.0	5268.0	5266.0	5270.0
1	20160701	90100270	42.0	5266.571429	-1.0	5268.0	5266.0	5270.0
2	20160701	90100518	72.0	5268.444444	1.0	5267.0	5266.0	5268.0
3	20160701	90100762	326.0	5270.000000	1.0	5268.0	5266.0	5270.0
4	20160701	90101019	6.0	5268.666667	-1.0	5270.0	5268.0	5272.0
...	...	...	...	...	...	...	...	...
397872	20160729	145858666	44.0	4996.000000	1.0	4995.0	4994.0	4996.0
397873	20160729	145858902	56.0	4996.000000	1.0	4995.0	4994.0	4996.0
397874	20160729	145859425	6.0	4995.333333	1.0	4995.0	4994.0	4996.0
397875	20160729	145859636	4.0	4996.000000	1.0	4995.0	4994.0	4996.0
397876	20160729	145859923	NaN	NaN	NaN	4995.0	4994.0	4996.0

397877 rows × 8 columns

In [ ]:
```
df2
```

Out[ ]:

	Date	Time	Size	VWAP	Sign	midQ	BP1	SP1
0	20160801	90100221	10.0	5084.000000	-1.0	5085.0	5084.0	5086.0
1	20160801	90100407	20.0	5086.000000	1.0	5085.0	5084.0	5086.0
2	20160801	90100745	16.0	5086.000000	1.0	5085.0	5084.0	5086.0
3	20160801	90100962	12.0	5085.666667	1.0	5085.0	5084.0	5086.0
4	20160801	90101246	28.0	5085.571429	1.0	5085.0	5084.0	5086.0
...	...	...	...	...	...	...	...	...
506306	20160831	145858815	44.0	5346.000000	-1.0	5347.0	5346.0	5348.0
506307	20160831	145859065	38.0	5347.263158	1.0	5347.0	5346.0	5348.0
506308	20160831	145859324	4.0	5346.000000	-1.0	5347.0	5346.0	5348.0
506309	20160831	145859572	4.0	5347.000000	0.0	5347.0	5346.0	5348.0
506310	20160831	145859792	NaN	NaN	NaN	5347.0	5346.0	5348.0

506311 rows × 8 columns

In [ ]:
```
#Concatenate the dataframes
df = pd.concat([df1, df2], ignore_index=True)
df
```

Out[ ]:

	Date	Time	Size	VWAP	Sign	midQ	BP1	SP1
0	20160701	90100020	48.0	5267.916667	-1.0	5268.0	5266.0	5270.0
1	20160701	90100270	42.0	5266.571429	-1.0	5268.0	5266.0	5270.0
2	20160701	90100518	72.0	5268.444444	1.0	5267.0	5266.0	5268.0
3	20160701	90100762	326.0	5270.000000	1.0	5268.0	5266.0	5270.0
4	20160701	90101019	6.0	5268.666667	-1.0	5270.0	5268.0	5272.0
...	...	...	...	...	...	...	...	...
904183	20160831	145858815	44.0	5346.000000	-1.0	5347.0	5346.0	5348.0
904184	20160831	145859065	38.0	5347.263158	1.0	5347.0	5346.0	5348.0
904185	20160831	145859324	4.0	5346.000000	-1.0	5347.0	5346.0	5348.0
904186	20160831	145859572	4.0	5347.000000	0.0	5347.0	5346.0	5348.0
904187	20160831	145859792	NaN	NaN	NaN	5347.0	5346.0	5348.0

904188 rows × 8 columns

$$\widetilde{R}_l = \langle(\hat{p}_{t+l} - m_t)\epsilon_t\rangle_{over\ t} \tag{3}$$

In [ ]:

```
# Function to calculate Rl_tilde for a given lag l
def calculate_Rl_tilde(df, l):
    vwap_t_l = df['VWAP'].shift(-l)   # Shift VWAP backwards by l
    mt = df['midQ']                   # mid-quote at time t
    epsilon_t = df['Sign']            # sign at time t

    Rl_tilde_values = (vwap_t_l - mt) * epsilon_t
    Rl_tilde_values.dropna(inplace=True)  # Drop NaN values resulting from the shif

    # Divide by the bid-ask spread (assuming bid and ask prices are available)
    bid_ask_spread = df['SP1'] - df['BP1']
    Rl_tilde_values /= bid_ask_spread

    return Rl_tilde_values.mean()

# Calculate Rl_tilde for 0 <= l <= 500
Rl_tilde_results = [calculate_Rl_tilde(df, l) for l in range(501)]
Rl_tilde_results
```

Out[ ]:
```
[0.45905368847382466,
 0.18321797564642547,
 0.18779777645341003,
 0.1944130755838694,
 0.19916333415297982,
 0.20221751592849224,
 0.20592058447937137,
 0.20761135839840048,
 0.20998503834816912,
 0.21199500031457494,
 0.212979188926931,
 0.213674865990605,
 0.2142056286494998,
 0.21503092223409012,
 0.21570706533534986,
 0.21563764826744494,
 0.21614310323071925,
 0.21593669732256948,
 0.21671344383320837,
 0.21681168524453728,
 0.21743760795230402,
 0.21785519243235096,
 0.21820162244697536,
 0.2176353065419987,
 0.21826919848945806,
 0.21764382868424384,
 0.2182839699454374,
 0.21747480400161695,
 0.21826266551998164,
 0.21778742836573978,
 0.2174977174970967,
 0.21797272787387786,
 0.2178733717461222,
 0.21783246180613064,
 0.21702862273813586,
 0.21715105786541214,
 0.21706819503431105,
 0.21697479411366266,
 0.21730629481210337,
 0.2165775257312885,
 0.21604421575795596,
 0.21541666962169784,
 0.21436871406235805,
 0.21428518816261335,
 0.21430269515987366,
 0.21487898036881042,
 0.21434591405812708,
 0.21480100106978614,
 0.21430489004105507,
 0.2137531300011323,
 0.21420758274989202,
 0.21375150613713215,
 0.21380003152149857,
 0.21345347478891438,
 0.21377937415895834,
 0.21371606954133096,
```

```
0.21351772356409324,
0.21271125446857708,
0.2123416910267314,
0.2122713789265727,
0.21309557299316173,
0.21301649749100768,
0.21237762860079074,
0.2121969335929431,
0.2114643811426892,
0.2119922931974323,
0.21177166640400513,
0.21091174261899276,
0.21087897148193246,
0.21130446386494564,
0.21106893019218906,
0.21097710559074892,
0.21019492288709654,
0.21007705211118305,
0.21060038918931656,
0.2096141153812966,
0.20976805263584,
0.2100231570140469,
0.21017119873541495,
0.20966124112446224,
0.2103083522147667,
0.20987031124966637,
0.21019484359285173,
0.210537478038849,
0.21024542287918413,
0.21062397371167405,
0.2099370485498098,
0.2107171030158973,
0.21066983551425608,
0.21059717631089608,
0.21110590913656968,
0.21199300669312793,
0.211623762136186,
0.21107974761045542,
0.21161655694051934,
0.2123803236107581,
0.21332961550207727,
0.2131331855512881,
0.21231996778124906,
0.21321133895635602,
0.21245263235320272,
0.21330099851516882,
0.2123629674357696,
0.21253875890039373,
0.21145020958516317,
0.21098584162840037,
0.21027561202041964,
0.20993010576937884,
0.2099888645844095,
0.21022459497758988,
0.21045772099661758,
0.21025344254082684,
```

```
0.2092094993291669,
0.20891143908457802,
0.20920175697238952,
0.20904177605244673,
0.20858464680396535,
0.2079963708010814,
0.2080774550985205,
0.20863313358681132,
0.20880969609389596,
0.2087188021424844,
0.20878516171846062,
0.20931714864305975,
0.2086308280023987,
0.20916788029765984,
0.2094465119498089,
0.20859076011818886,
0.20948714973993554,
0.2105126389859239,
0.20935360814511658,
0.20944955928489703,
0.20972098797707828,
0.20880455915541932,
0.20866694731407584,
0.2076214920224507,
0.20802361002679579,
0.20769103652133128,
0.20798113092790785,
0.20730657699781874,
0.2070880786158019,
0.2070048806544656,
0.2066134213392393,
0.20719239047195892,
0.2070384484479215,
0.20674071309636508,
0.2076215051955823,
0.2075928462031799,
0.20782113469319807,
0.20760093231212684,
0.20662951791425868,
0.2068648627722376,
0.2074777278694325,
0.2076209533180313,
0.20816860165444337,
0.20745076083095257,
0.206766679786503,
0.20657548334619696,
0.20662236116329138,
0.20753144604977336,
0.20701712077725581,
0.20729090199746553,
0.20752059814454865,
0.20749310617241556,
0.2072852921418772,
0.20780001457720604,
0.20742143179398487,
0.20773504594431788,
```

```
0.20698176191350962,
0.2069645506723673,
0.20774300650429908,
0.20788468583925646,
0.20713290158669595,
0.20731460930449166,
0.20711669433474852,
0.20774823934332964,
0.20634786399643096,
0.2062635412250434,
0.20711945532980286,
0.20670435754452474,
0.20714220060257868,
0.20687576085353793,
0.20693207091908003,
0.2062361874383547,
0.20612894131428874,
0.20683023707366627,
0.20642526839114292,
0.20645070588162895,
0.20684857556594183,
0.20632952311079697,
0.2055673951541789,
0.2064186596522641,
0.2058341086455587,
0.20605148295069386,
0.20646586954877508,
0.20653578172248555,
0.20605345512829326,
0.20593247091259106,
0.20630909429587627,
0.2060779242039404,
0.20634657215322116,
0.20699418272203926,
0.20679925781223302,
0.20653956152454572,
0.20729248246727044,
0.20766825712676285,
0.20676524676511,
0.2069287284644082,
0.20643062934239997,
0.20687216869286315,
0.20674111133036735,
0.2065749942818388,
0.20723184068541858,
0.2067884051510963,
0.20669096260416894,
0.2074973756637029,
0.2078537495760744,
0.20861135167758552,
0.2083502916819167,
0.2090423138876537,
0.2085816321742316,
0.2087185011135052,
0.20880167805804578,
0.208391641894962,
```

```
0.20826965722719304,
0.20865649014036153,
0.20887110991488456,
0.2086688866389234,
0.2078569666897041,
0.2090727085277837,
0.20832172535837276,
0.20878712047380296,
0.20917255756452507,
0.20946731860517434,
0.20978401614382003,
0.2093355311619854,
0.20889251344269544,
0.2098246748788927,
0.20901615918115785,
0.20904387714360365,
0.20861221373181454,
0.20773037910822897,
0.2077440092603489,
0.20824346718107192,
0.20835886438096918,
0.20837301297788333,
0.2085331418109894,
0.20882504299889162,
0.208887792438439,
0.20842966235350344,
0.20904765934109193,
0.20915469592722996,
0.20850684227494415,
0.20762351796686193,
0.20727608874568648,
0.20849116579399807,
0.2081030844675283,
0.2084971751420651,
0.20882144704778574,
0.20831655467733698,
0.20859773595287773,
0.20912562109430743,
0.2093284859219902,
0.20938340918533177,
0.20854192939237276,
0.20881351500281087,
0.20878148707002867,
0.2090620595601086,
0.20861219932523867,
0.2088689470787359,
0.20890827672038778,
0.20909675674078115,
0.20849018681451353,
0.2079515919699084,
0.20847513955636862,
0.20851760609653397,
0.20807161130127821,
0.20758376684137422,
0.20714361564971084,
0.20656330994684757,
```

```
0.207399675389312,
0.20711949650128536,
0.2077367877676372,
0.20772028683755905,
0.20709256266073417,
0.2076850347285127,
0.20762305025471042,
0.20841361406662334,
0.20888367170199162,
0.2088822517800586,
0.20859321161668237,
0.20786438398363377,
0.2086768328878253,
0.20875601636111762,
0.20847213735687778,
0.20789957940623907,
0.20834100485998758,
0.2074563460125855,
0.20771754045469856,
0.2078253193006404,
0.20795031022532479,
0.20847681128781254,
0.20847849075446306,
0.20740670914698797,
0.20753748958037932,
0.2077270604359467,
0.20713446862448312,
0.20723234215929281,
0.20768228649050968,
0.2073207004094523,
0.20719035774830308,
0.2078428142030451,
0.2078241896535859,
0.20744948738706645,
0.20823287816676556,
0.20795221347742607,
0.2073182797860471,
0.20842042277651185,
0.20814667526030514,
0.2080485425230049,
0.20776722505282996,
0.20808549961161277,
0.20807999442044686,
0.20785676075046375,
0.20716687527341593,
0.20799901007371083,
0.20816424949977924,
0.20867199362846864,
0.20865721837508164,
0.20868944940046089,
0.20950149970421197,
0.2090030238707639,
0.2094944227420218,
0.20903328686830444,
0.20973688166070842,
0.20890515660301803,
```

```
0.20866727655885134,
0.20879077005254332,
0.20883258969768373,
0.20921498163473934,
0.20998670957332682,
0.20959903776917058,
0.20849767741775171,
0.2095155682312217,
0.20977953658609422,
0.20935579922578773,
0.20929018889421577,
0.20939110350160076,
0.21007649232180042,
0.2096590891710536,
0.21021234677086056,
0.20963396422004088,
0.2104255527250405,
0.21059685646217752,
0.21045320712847404,
0.2108861092789094,
0.21012684688650982,
0.2101529682834802,
0.21066188556256182,
0.2109328353199838,
0.21092494120534427,
0.21057957089045073,
0.21111570603676705,
0.21100611433312796,
0.2109468933820881,
0.21093072732213855,
0.2121376562251901,
0.21272239846771643,
0.21311543074422196,
0.2124130994398788,
0.213008951839565,
0.21238380731014733,
0.21347936786031718,
0.21324666117285296,
0.21344012051283986,
0.21387117735892666,
0.21396683349582693,
0.2144988543644384,
0.21462742017099604,
0.21443990988243997,
0.21368884770618787,
0.21444513918178323,
0.21426091936894684,
0.21408222777665592,
0.21383222402446356,
0.21422312326321502,
0.21450914297011822,
0.21480227499600227,
0.21497497071315688,
0.2147977938625524,
0.21511089662677724,
0.21547105641428613,
```

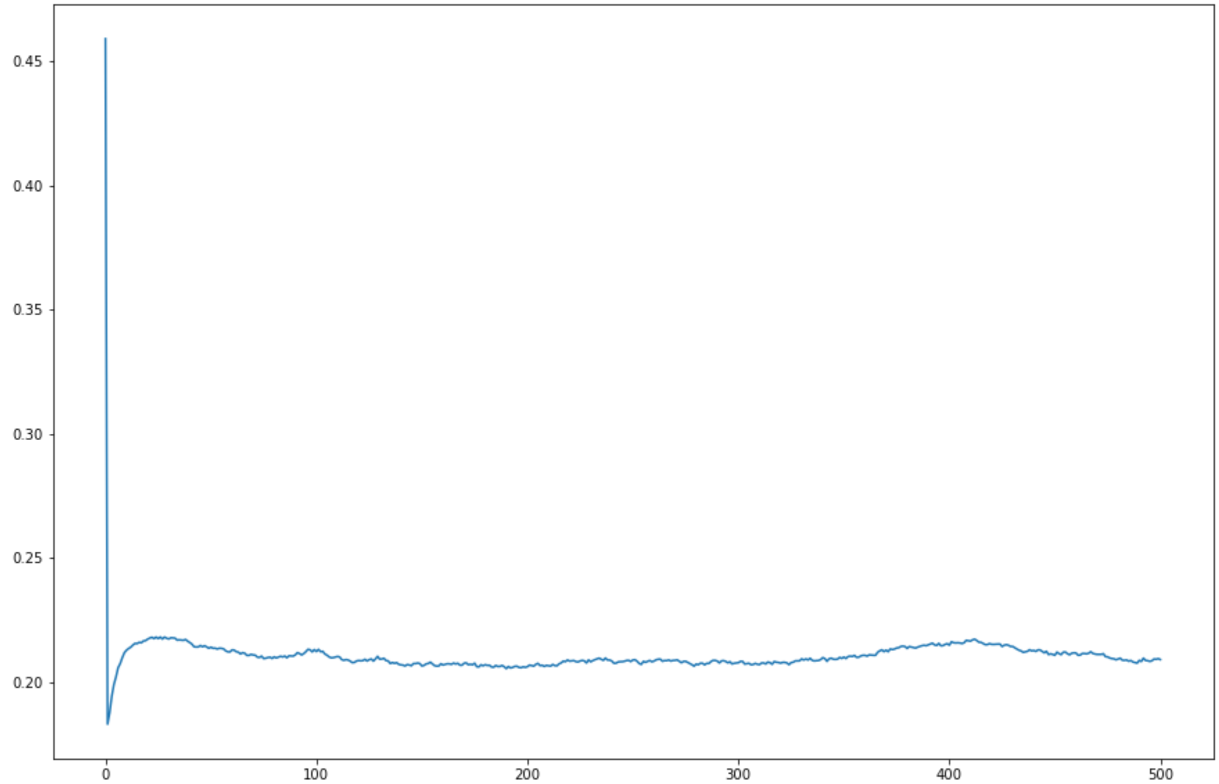
```
0.2158094435502924,
0.2151127022672172,
0.21488724623889,
0.21573697539990525,
0.2147654150044285,
0.21484138796832572,
0.2153269591195433,
0.2156034482244217,
0.21500505242797602,
0.2163868575775403,
0.21598873044283146,
0.21589742839678722,
0.21597226662856905,
0.21588908215525007,
0.21567660477554992,
0.2159859806870109,
0.21695655078612805,
0.21681248753840543,
0.21661877084015205,
0.21715882622239682,
0.21742433696012792,
0.21686944823680082,
0.21620029058159085,
0.21614608091750062,
0.21567590843990772,
0.2152282472322986,
0.21523661357723778,
0.21563426862428842,
0.21536287122392966,
0.21532686318495042,
0.2153959570507989,
0.21544649503929802,
0.21545315925392752,
0.21446638913862148,
0.2149950389774955,
0.21519556739305645,
0.2149898392712325,
0.21448829007206877,
0.21449198490187016,
0.21395446447509353,
0.21345546238169277,
0.21297615127770275,
0.2127016048608244,
0.2121032662461219,
0.21229279862864178,
0.21253309305383325,
0.21315579711454075,
0.21271052960856038,
0.21287292065731545,
0.21249344746121845,
0.21309875217443147,
0.21307051584696585,
0.21250679734243416,
0.21193207053104493,
0.21254449702426068,
0.2112386284610691,
```

```
 0.21151636363921048,
 0.21127794476286585,
 0.21094060835088646,
 0.21220707889414087,
 0.2115937991996909,
 0.21119821451709797,
 0.21217001788651452,
 0.21225207217374334,
 0.2115469860110044,
 0.21114352587410534,
 0.21177043283989794,
 0.2119237935396691,
 0.21186770374593403,
 0.21095841060957488,
 0.21101553151494165,
 0.2116494908904276,
 0.21165630542570646,
 0.21151186697049174,
 0.21173519233879481,
 0.21236643666846053,
 0.21169978159838998,
 0.21144521356875967,
 0.2112990970567669,
 0.21128888442300525,
 0.2113048420536382,
 0.21155107654546063,
 0.21039047071601066,
 0.21030170066372614,
 0.20984923751756912,
 0.20961754753779285,
 0.20951365984892287,
 0.20912029004645363,
 0.2094477992303557,
 0.20975847416358914,
 0.2089583188120645,
 0.2089141020593893,
 0.20906018978566138,
 0.20855364107974095,
 0.2088190223908828,
 0.2082530290080894,
 0.2080208057263671,
 0.20777413405409628,
 0.20887197696336832,
 0.2084529740950369,
 0.20979425069884983,
 0.20887514115132505,
 0.20875236253683538,
 0.20849796918560454,
 0.2087038297246002,
 0.20943924200384118,
 0.20931573720752822,
 0.209581680304603,
 0.20922392065197473]
```

In [ ]:

```
#PLot Rl_tilde_results against l and change the figure size
plt.figure(figsize=(15,10))
```

```
plt.plot(Rl_tilde_results)
plt.show()
```



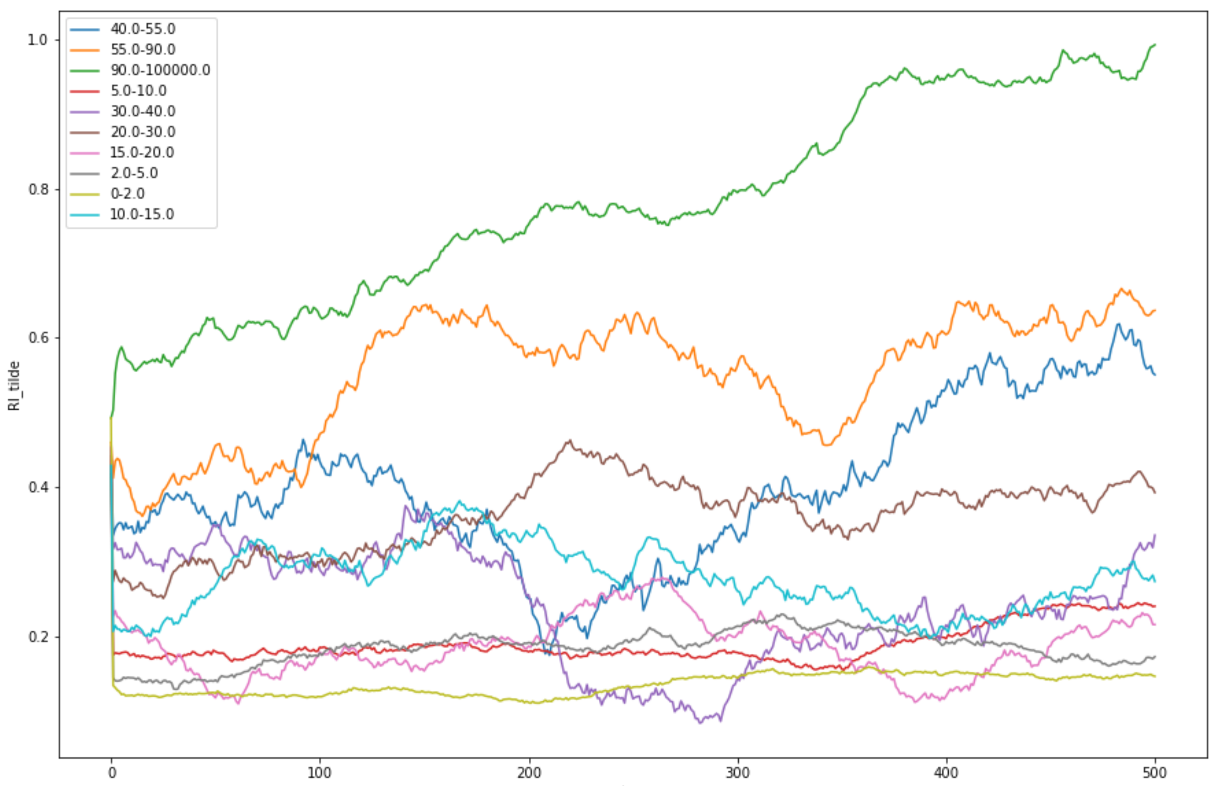
---

6. **Question 3 (25 points)**: With the data provided with this assignment, construct $\widetilde{R_l}\Big|_V$ for $0 \le l \le 500$ as defined in equation (3) for trades in different groups of trade sizes. That is, if we label all trades that have sizes that $v_i < V_i \le v_{i+1}$ as group i, calculate $\widetilde{R_l}\Big|_{v_i<V_i<v_{i+1}}$ for $0 \le l \le 500$ as defined in equation (3) for all the anchoring trades within group i. Note that any trade can be an anchoring trade, except the last few ones in a time series depending on the value of l. Comment on your findings from this analysis, especially on how the response function depends on trade sizes. In this assignment, we define: $v_1 = 0, v_2 = 2, v_3 = 5, v_4 = 10, v_5 = 15, v_6 = 20, v_7 = 30, v_8 = 40, v_9 = 55, v_{10} = 90, v_{11} = 100000.$

In [ ]:
```
def calculate_average_Rl_tilde_by_size(df, max_l):
    # Define trade size categories
    size_categories = [0, 2.0, 5.0, 10.0, 15.0, 20.0, 30.0, 40.0, 55.0, 90.0, 10000
    #add 1 to each element in the list
    size_categories = [x+0.1 for x in size_categories]
    size_categories

    # Create a new column to represent trade size groups
    df['TradeSizeGroup'] = pd.cut(df['Size'], bins=size_categories, labels=False, r

    average_Rl_tilde_results_by_size = {}

    for size_group in df['TradeSizeGroup'].unique():
```

```
        group_df = df[df['TradeSizeGroup'] == size_group]

        # Calculate Rl_tilde for each l in the range [0, max_l]
        Rl_tilde_results = [calculate_Rl_tilde(group_df, l) for l in range(max_l +

        # Save the results for each group as a dictionary entry
        average_Rl_tilde_results_by_size[size_group] = Rl_tilde_results

    return average_Rl_tilde_results_by_size

# Example usage:
max_l = 500
average_Rl_tilde_results_by_size = calculate_average_Rl_tilde_by_size(df, max_l)
```

In [ ]:
```
size_categories = [0, 2.0, 5.0, 10.0, 15.0, 20.0, 30.0, 40.0, 55.0, 90.0, 100000.0]
create_bin_tags = [str(x) + '-' + str(y) for x, y in zip(size_categories[:-1], size
plt.figure(figsize=(15,10))
#plot the results for each bin in the same figure
for size_group in average_Rl_tilde_results_by_size:
    #if size_group is nan then skip
    if np.isnan(size_group):
        continue
    plt.plot(average_Rl_tilde_results_by_size[size_group], label=create_bin_tags[in

plt.xlabel('Lag')
plt.ylabel('Rl_tilde')
plt.legend()
plt.show()
```

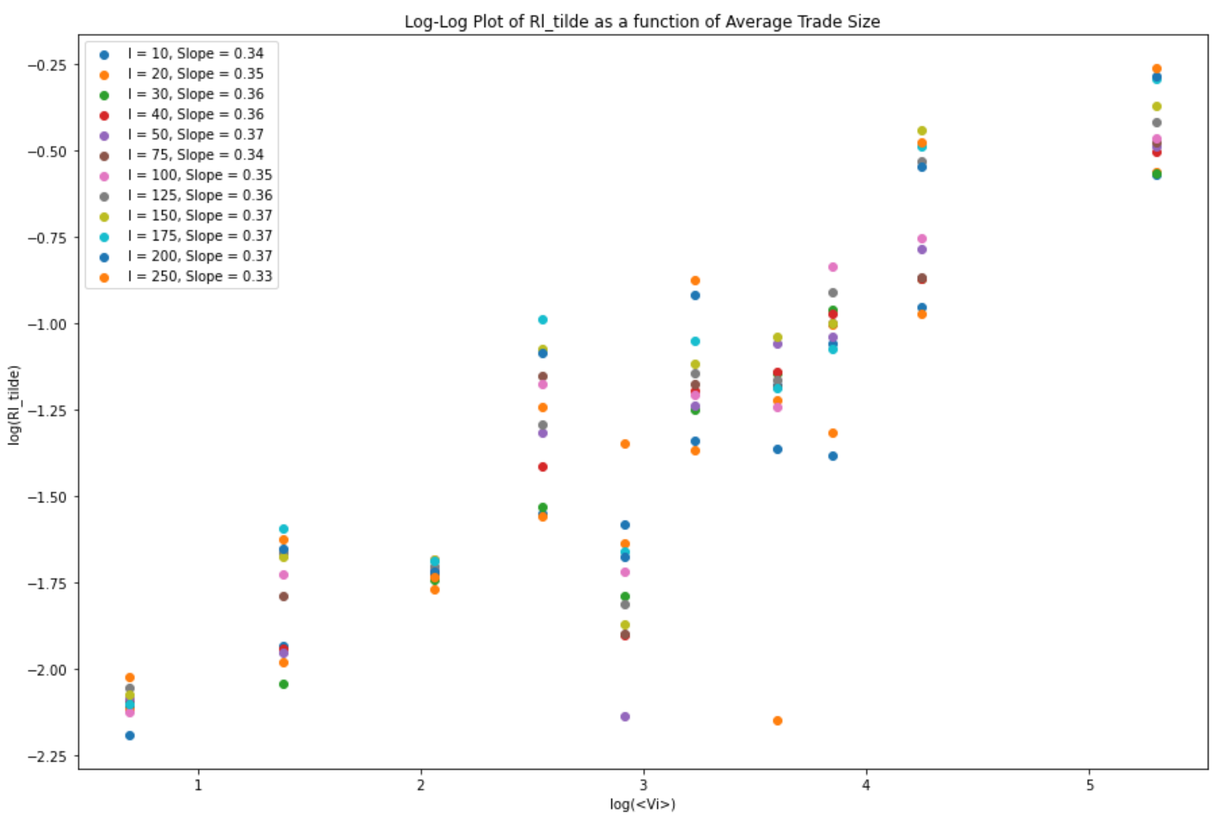


> We can note that higher the size (volume) of a trade cluster the higher is the impact and the response of the market to the trade cluster.When a large trade is executed, it can lead to price movements, affecting the VWAP and mid-quotes. The larger the trade size, the more likely it is to cause noticeable market impact.
>
> Larger trades may also have a more pronounced impact on the bid-ask spread. The bid-ask spread is use in calculating response functions. If larger trades widen the spread or cause temporary imbalances in supply and demand, the response function may show a higher value.

7. **Question 4 (25 points)**: For $l = 10, 20, 30, 40, 50, 75, 100, 125, 150, 175, 200, 250$, plot $\log(\widetilde{R}_l|_{v_i < V_i < v_{i+1}})$ as a function of $\log(\langle V_i \rangle)$ and fit the data into a straight line. Compare the slopes of different straight lines for different l. $\langle V_i \rangle$ is the average of trade sizes of all trades in group i.

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

# Function to fit a line to the data and return the slope
def fit_line(x, y):
    slope, intercept, r_value, p_value, std_err = linregress(x, y)
    return slope

# Function to calculate log(Rl_tilde) for a given lag l and trade size group
def calculate_log_Rl_tilde(df, l):
    vwap_t_l = df['VWAP'].shift(-l)
    mt = df['midQ']
    epsilon_t = df['Sign']

    Rl_tilde_values = (vwap_t_l - mt) * epsilon_t
    Rl_tilde_values.dropna(inplace=True)

    bid_ask_spread = df['SP1'] - df['BP1']
    Rl_tilde_values /= bid_ask_spread

    return np.log(Rl_tilde_values.mean())

# Function to calculate log(<Vi>) for a given trade size group
def calculate_log_average_trade_size(df):
    return np.log(df['Size'].mean())

# Plotting
lags = [10, 20, 30, 40, 50, 75, 100, 125, 150, 175, 200, 250]
size_categories = [0, 2.0, 5.0, 10.0, 15.0, 20.0, 30.0, 40.0, 55.0, 90.0, 100000.0]

plt.figure(figsize=(15, 10))
```

```python
for l in lags:
    log_Rl_tilde_values = []
    log_average_trade_size_values = []

    for size_group in range(len(size_categories) - 1):
        group_df = df[(df['Size'] > size_categories[size_group]) & (df['Size'] <= s
        log_Rl_tilde = calculate_log_Rl_tilde(group_df, l)
        log_average_trade_size = calculate_log_average_trade_size(group_df)

        log_Rl_tilde_values.append(log_Rl_tilde)
        log_average_trade_size_values.append(log_average_trade_size)

    # Fit a line to the data
    slope = fit_line(log_average_trade_size_values, log_Rl_tilde_values)

    # Plot the data points
    plt.scatter(log_average_trade_size_values, log_Rl_tilde_values, label=f'l = {l}
plt.xlabel('log(<Vi>)')
plt.ylabel('log(Rl_tilde)')
plt.legend()
plt.title('Log-Log Plot of Rl_tilde as a function of Average Trade Size')
plt.show()
```

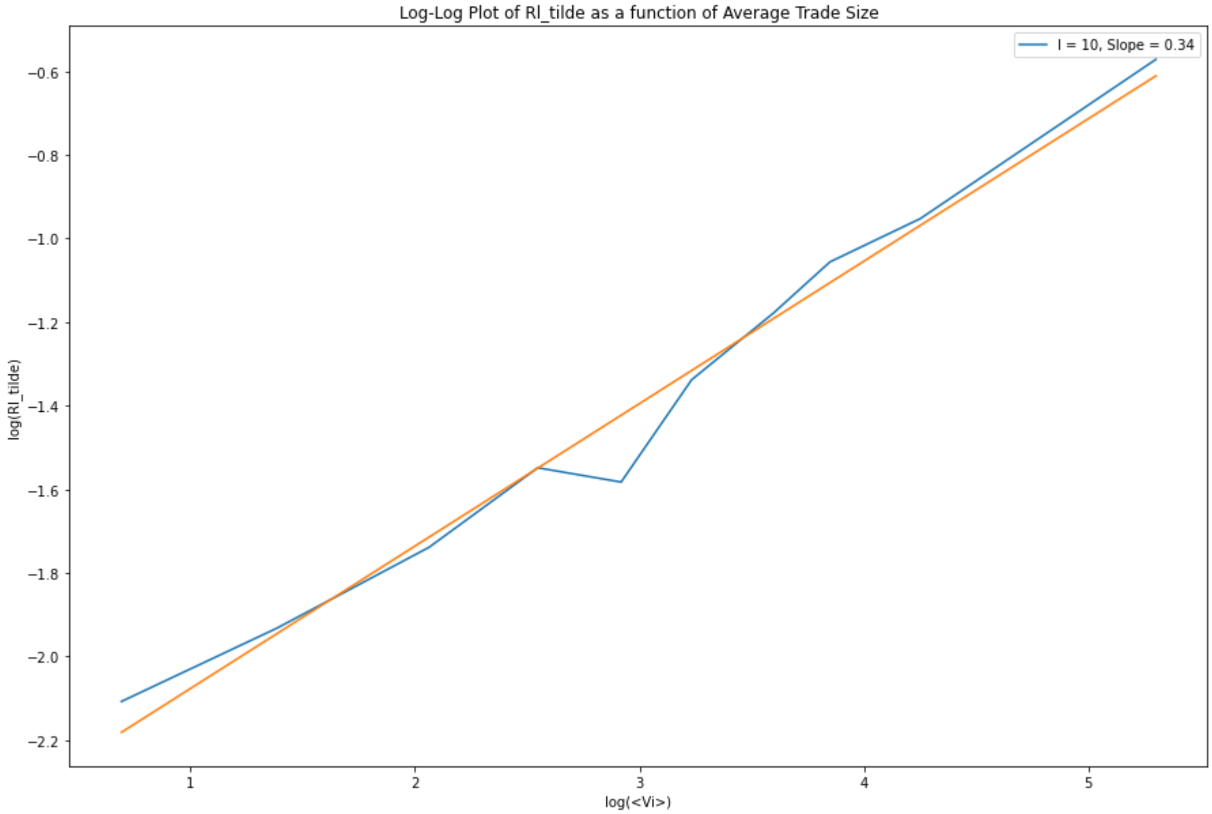


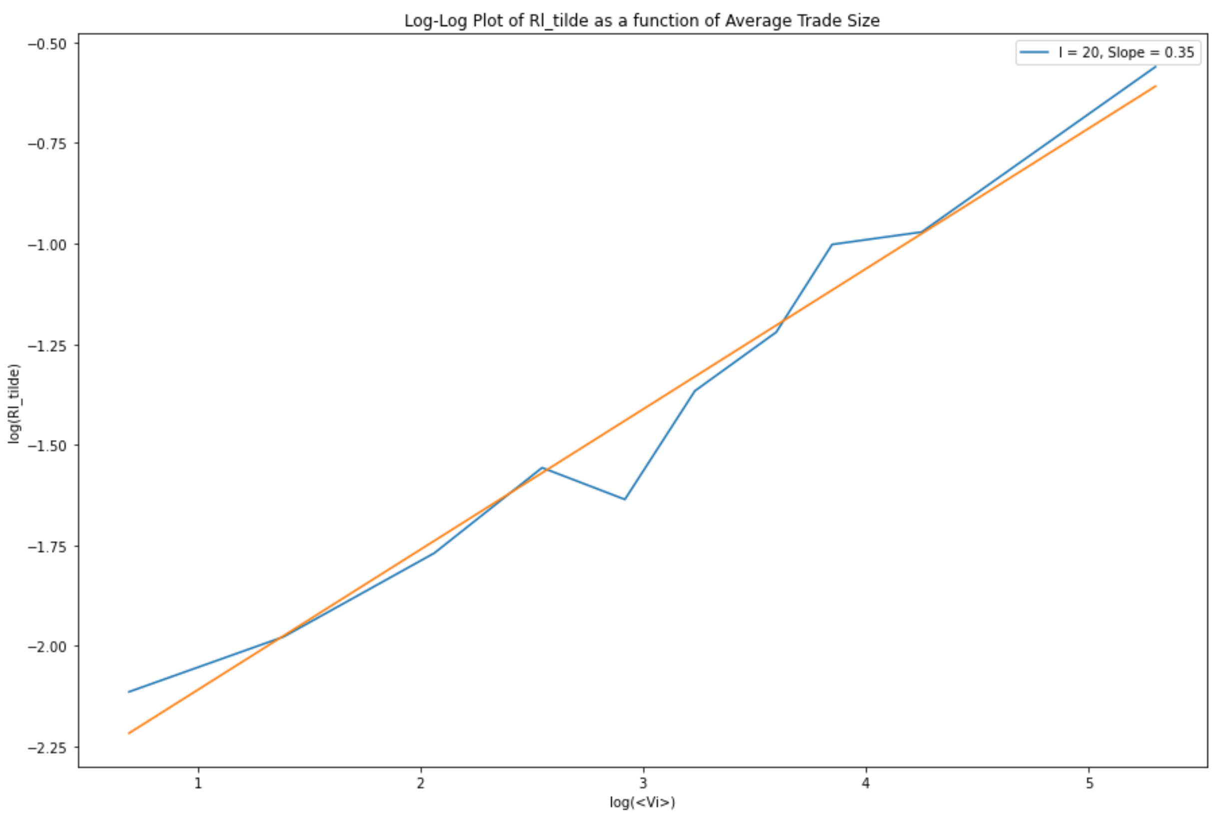
In [ ]:

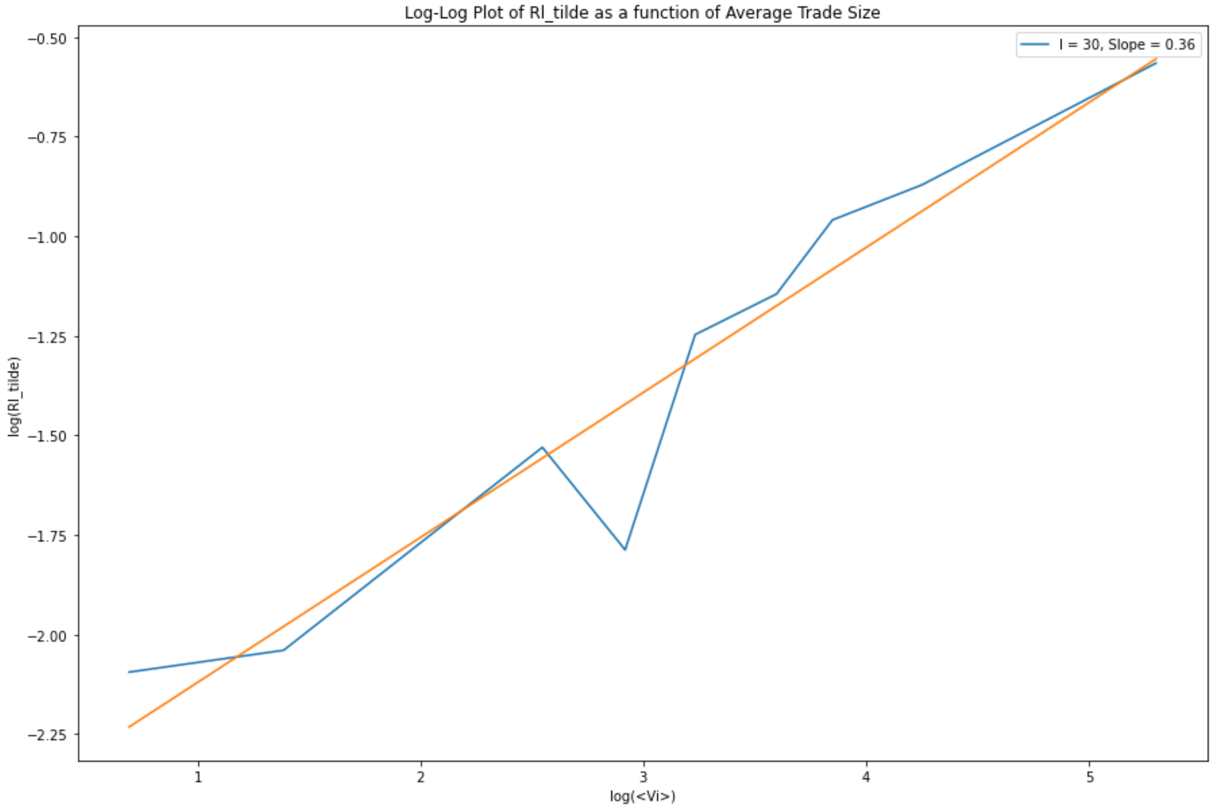
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

# Function to fit a line to the data and return the slope
def fit_line(x, y):
```

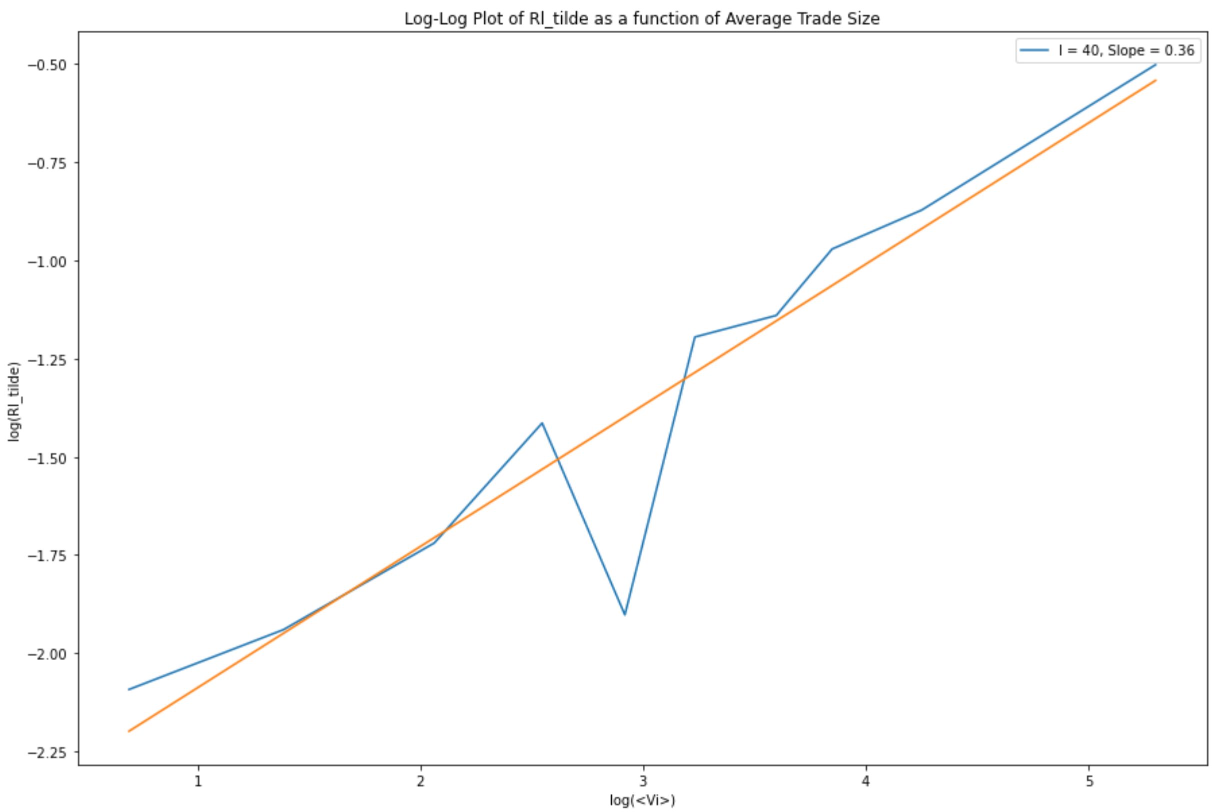
```
    slope, intercept, r_value, p_value, std_err = linregress(x, y)
    return slope

# Function to calculate log(Rl_tilde) for a given lag l and trade size group
def calculate_log_Rl_tilde(df, l):
    vwap_t_l = df['VWAP'].shift(-l)
    mt = df['midQ']
    epsilon_t = df['Sign']

    Rl_tilde_values = (vwap_t_l - mt) * epsilon_t
    Rl_tilde_values.dropna(inplace=True)

    bid_ask_spread = df['SP1'] - df['BP1']
    Rl_tilde_values /= bid_ask_spread

    return np.log(Rl_tilde_values.mean())

# Function to calculate log(<Vi>) for a given trade size group
def calculate_log_average_trade_size(df):
    return np.log(df['Size'].mean())

# Plotting
lags = [10, 20, 30, 40, 50, 75, 100, 125, 150, 175, 200, 250]
size_categories = [0, 2.0, 5.0, 10.0, 15.0, 20.0, 30.0, 40.0, 55.0, 90.0, 100000.0]

plt.figure(figsize=(15, 10))


for l in lags:
    log_Rl_tilde_values = []
    log_average_trade_size_values = []

    for size_group in range(len(size_categories) - 1):
        group_df = df[(df['Size'] > size_categories[size_group]) & (df['Size'] <= s
        log_Rl_tilde = calculate_log_Rl_tilde(group_df, l)
        log_average_trade_size = calculate_log_average_trade_size(group_df)

        log_Rl_tilde_values.append(log_Rl_tilde)
        log_average_trade_size_values.append(log_average_trade_size)

    # Fit a line to the data
    slope = fit_line(log_average_trade_size_values, log_Rl_tilde_values)

    # Plot the data points
    #plt.scatter(log_average_trade_size_values, log_Rl_tilde_values, label=f'l = {l
    plt.figure(figsize=(15, 10))
    plt.plot(log_average_trade_size_values, log_Rl_tilde_values, label=f'l = {l}, S
    #plot a line of best fit
    plt.plot(np.unique(log_average_trade_size_values), np.poly1d(np.polyfit(log_ave
    plt.xlabel('log(<Vi>)')
    plt.ylabel('log(Rl_tilde)')
    plt.legend()
    plt.title('Log-Log Plot of Rl_tilde as a function of Average Trade Size')
    plt.show()
```

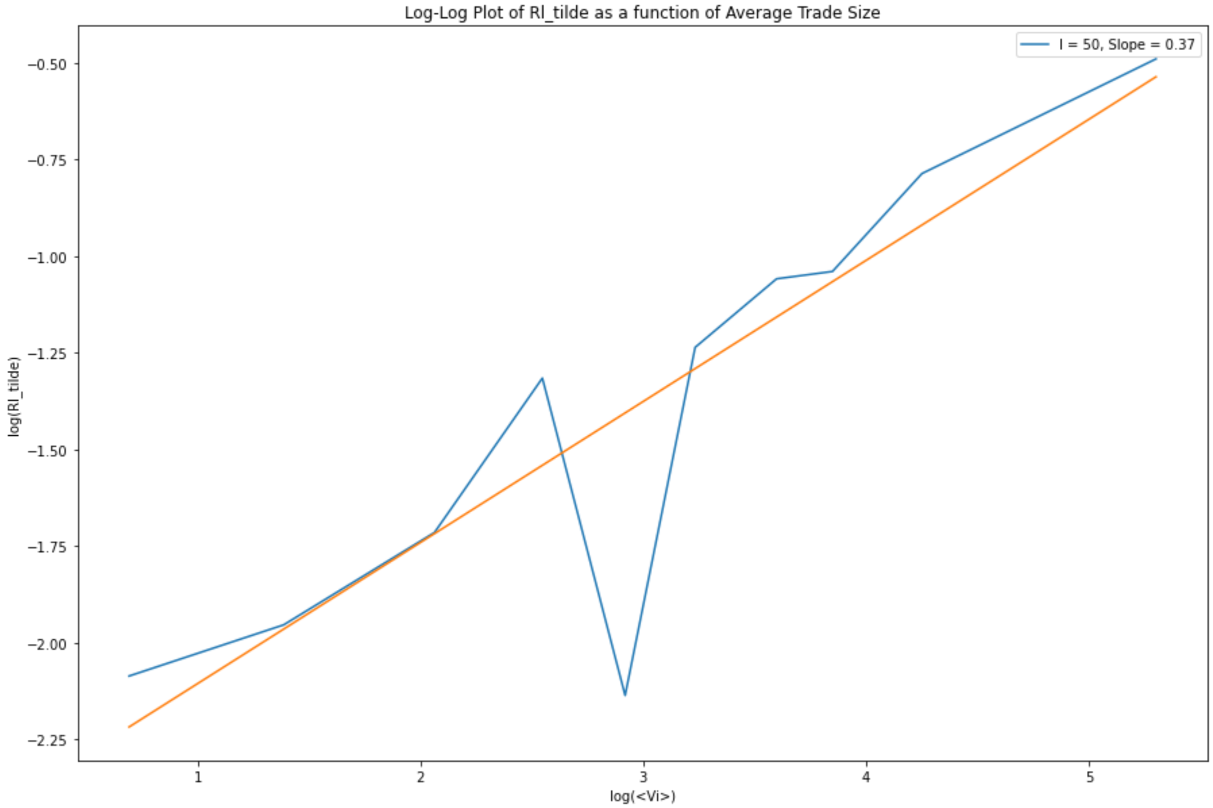
```
<Figure size 1080x720 with 0 Axes>
```

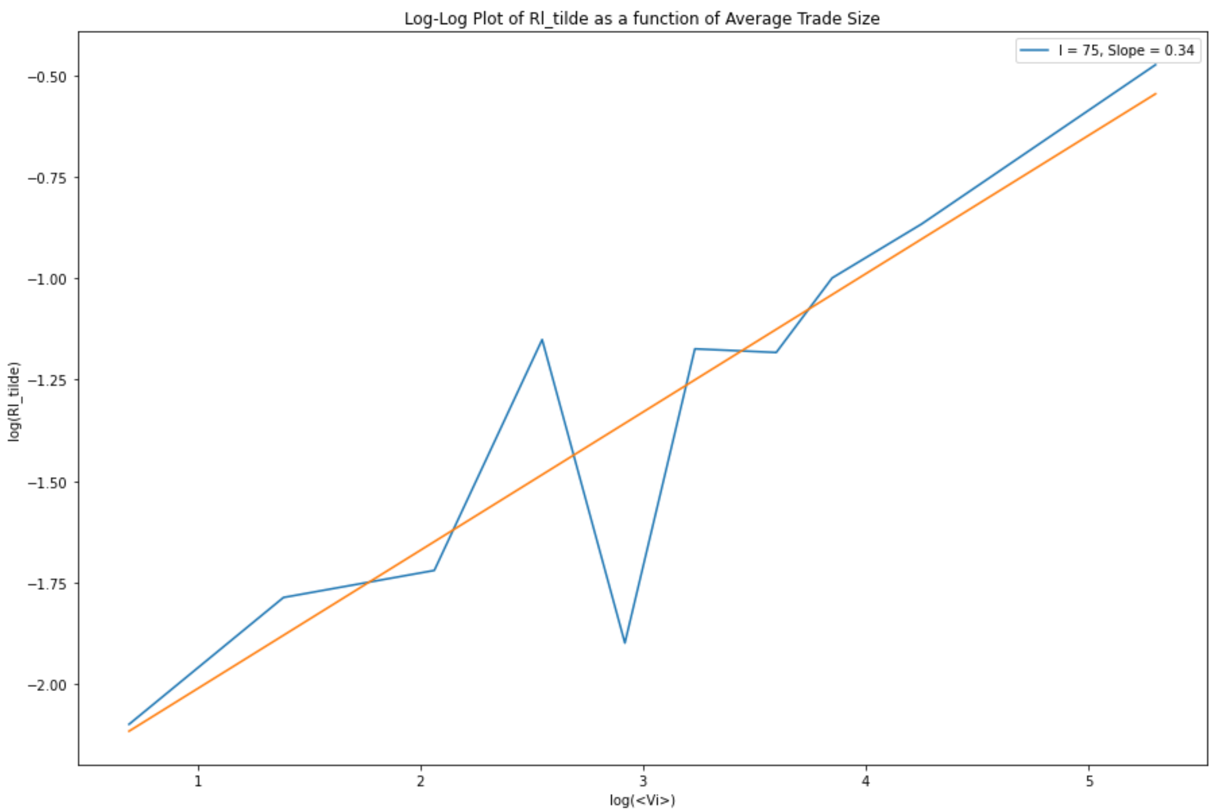

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 10, Slope = 0.34
log(Rl_tilde)
log(<Vi>)

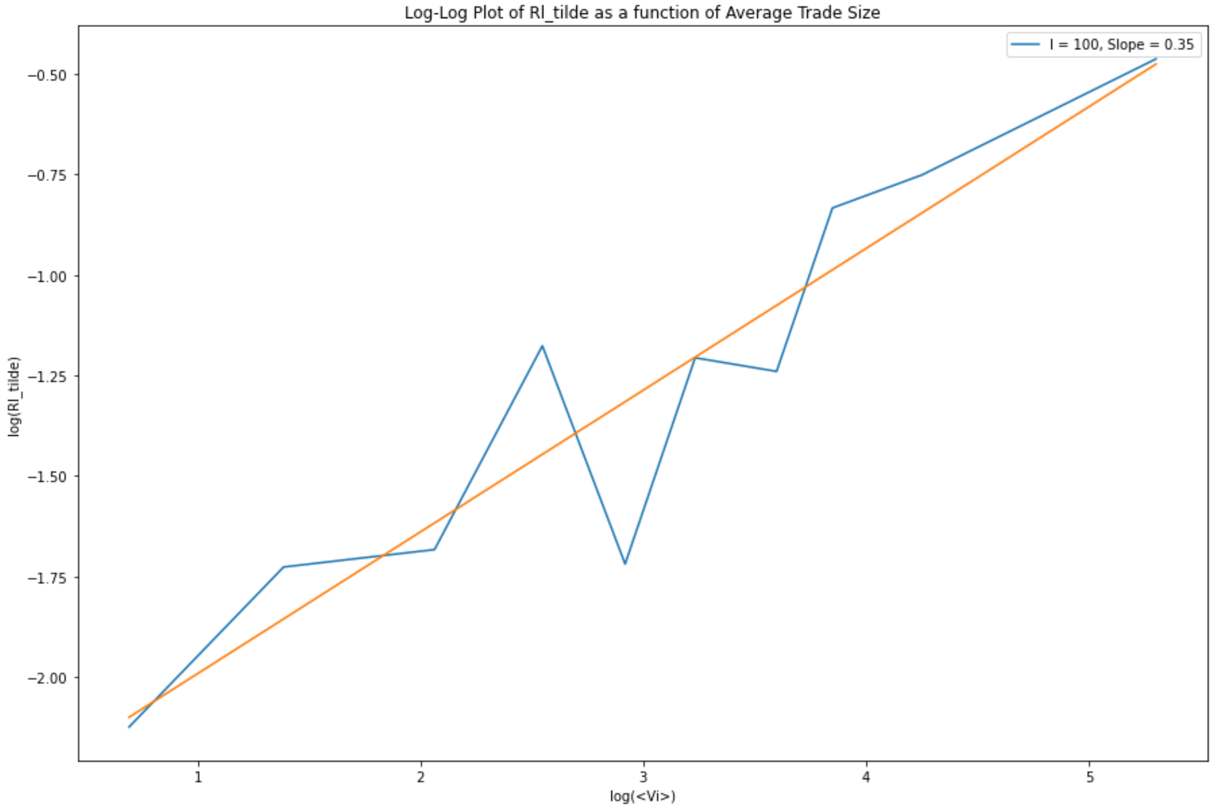

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 20, Slope = 0.35
log(Rl_tilde)
log(<Vi>)

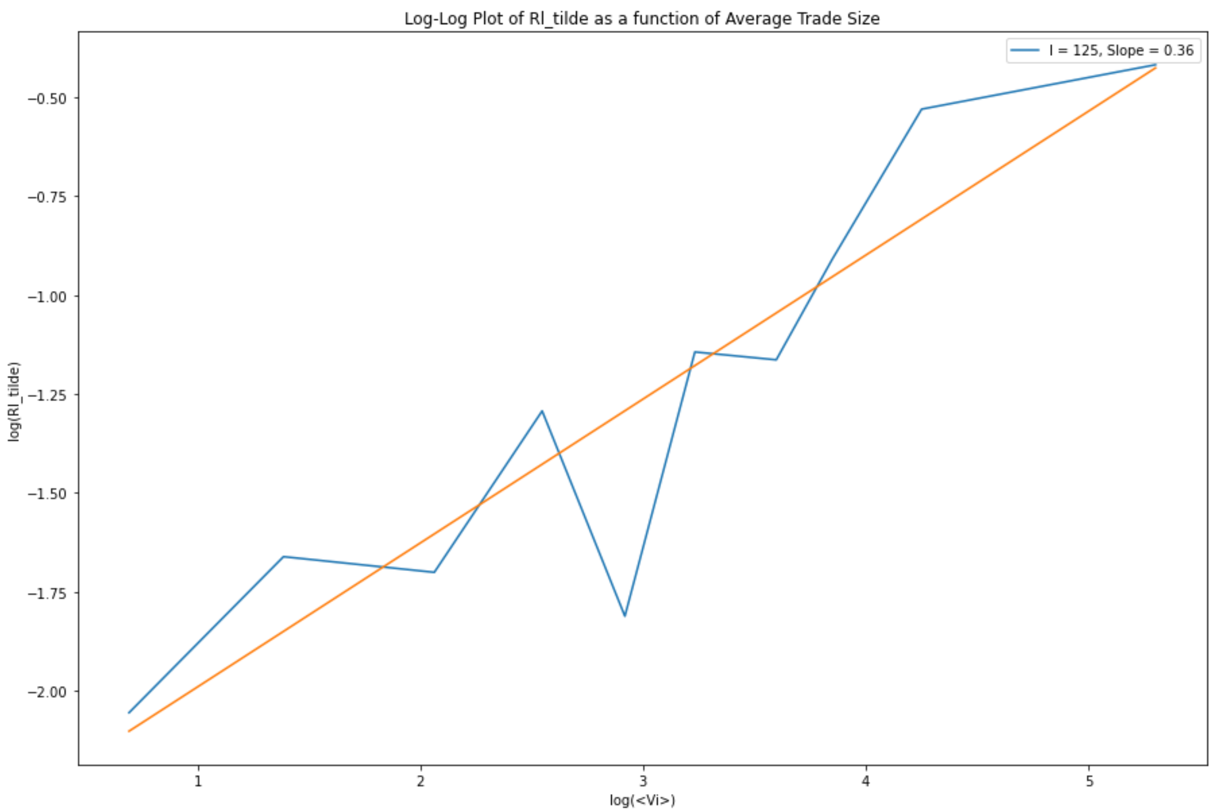

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 30, Slope = 0.36
log(Rl_tilde)
log(<Vi>)

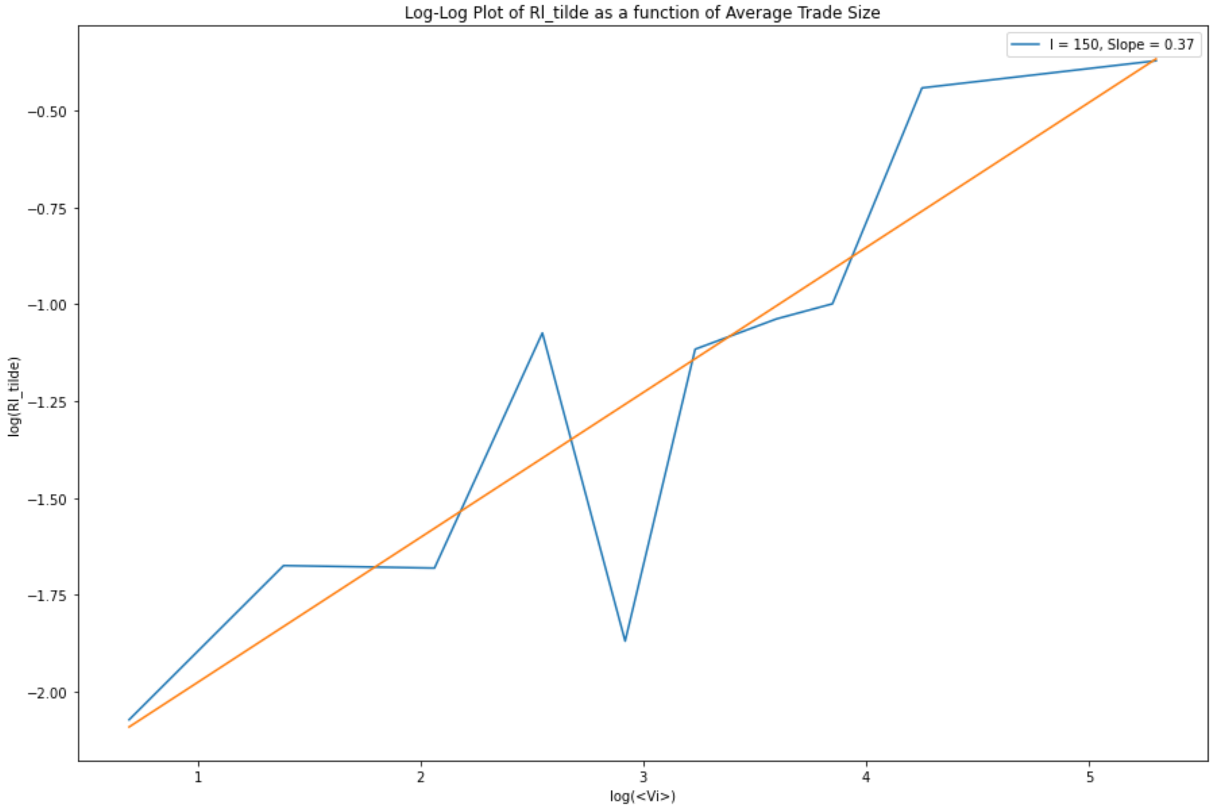

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 40, Slope = 0.36
log(Rl_tilde)
log(<Vi>)

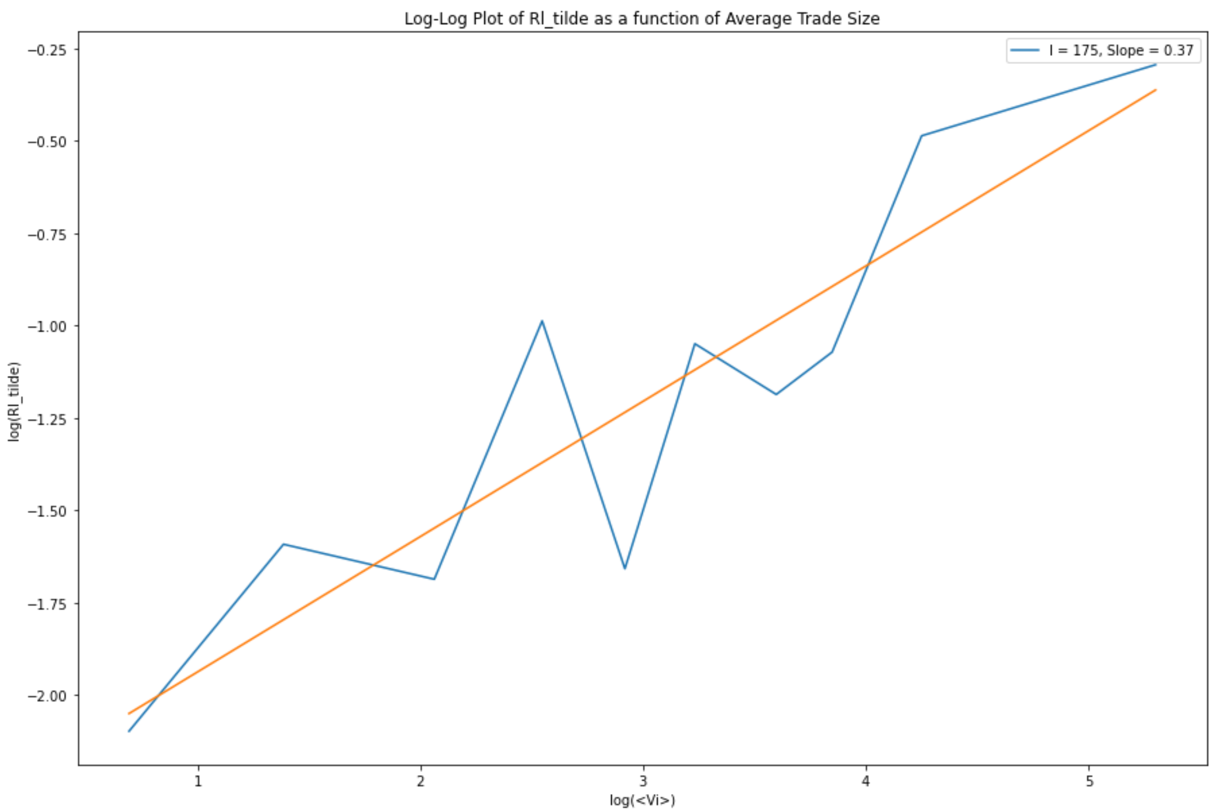

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 50, Slope = 0.37
log(Rl_tilde)
log(<Vi>)

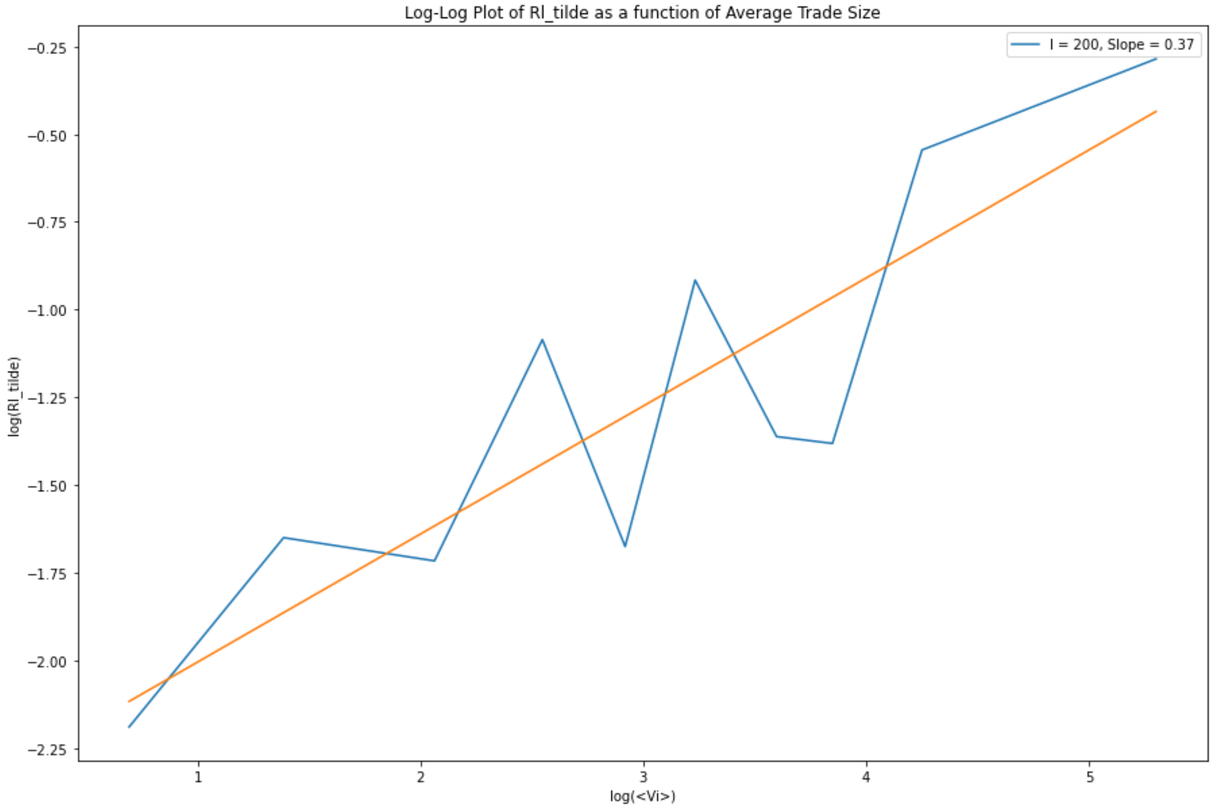

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 75, Slope = 0.34
log(Rl_tilde)
log(<Vi>)

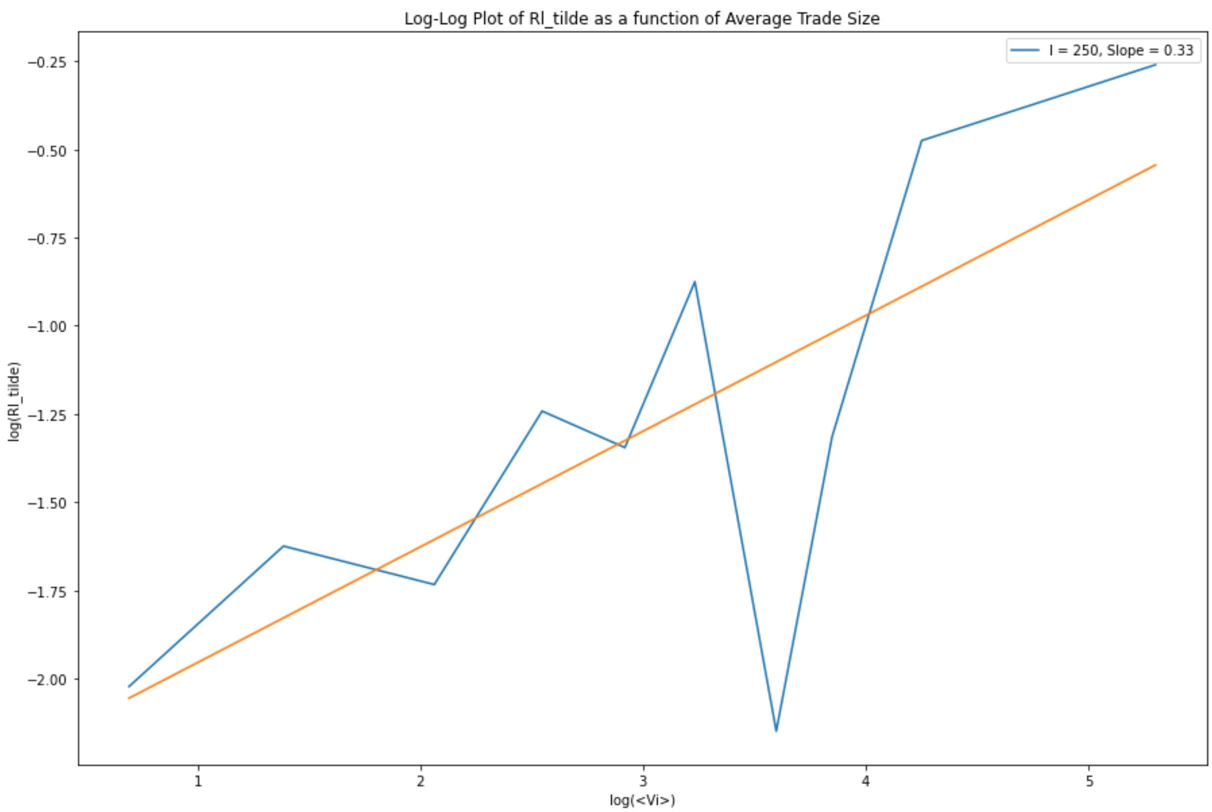

Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 100, Slope = 0.35
log(Rl_tilde)
log(<Vi>)
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
1
2
3
4
5


Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 125, Slope = 0.36
log(Rl_tilde)
log(<Vi>)
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
1
2
3
4
5


Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 150, Slope = 0.37
log(Rl_tilde)
log(<Vi>)
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
1
2
3
4
5


Log-Log Plot of Rl_tilde as a function of Average Trade Size
I = 175, Slope = 0.37
log(Rl_tilde)
log(<Vi>)
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
1
2
3
4
5





> Comparing the slopes we see a bit of consistency of the slopes not being too far apart in the range of 0.33 to 0.37. But, more interestingly the R^2 values reduce as we increase the size of the lag. We see more variantion with the log (size) with a higher lag compared to shorter lags.