Project 3

Task 1

Members: Lars Olav Thorbjørnsen, Stein Are Årsnes og Sanjai Vijayaratnam

Abstract

Data preprocessing and feature selection are essential steps in machine learning(ML) to ensure model accuracy and reliability. In this task, we take the raw log data, containing shear wave velocity (Vs), density (DEN), neutron porosity (NEU) and compressional wave velocity (Vp), clean it and prepare it for analysis. We start with checking that all data is of the float type before moving on to missing values, duplicates, outliers and conducting a correlation analysis. After removing missing values, duplicates and most of the outliers, we choose too keep all features since they are all above the 0.5 threshold. In the end we do some filtering and display the results. We have now laid a strong foundation for ML modeling, improving the reliability of Vp estimation from the log data.

Introduction

Effective data preprocessing and feature selection are important in building accurate ML models. In this project, raw data logs with attributes like Vs, DEN and NEU needs to be cleaned and prepared before being used to estimate Vp. Raw data could be incomplete, redundant or noisy, by data preprocessing these issues can be fixed [1]. This task aimed to enhance data quality and select meaningful predictors to ensure the effectiveness of ML models.

Task 1: Data Pre-processing

In [15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_excel('ProjectData2024.xlsx')
df.info()
df.head(), df.tail(20)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1195 entries, 0 to 1194
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Vs      1187 non-null   float64
 1   DEN     1185 non-null   float64
 2   NEU     1184 non-null   float64
 3   Vp      1182 non-null   float64
dtypes: float64(4)
memory usage: 37.5 KB
```

Out[15]:
```
(          Vs     DEN     NEU        Vp
 0  1.676857  2.3767  0.2759  3.045533
 1  1.677172  2.2101  0.2524  2.974779
 2  1.676252  2.1419  0.2591  2.881411
 3  1.677030  2.1660  0.2589  2.930981
 4  1.684534  2.1193  0.2596  2.910094,
             Vs     DEN     NEU        Vp
 1175  1.830254  2.4994  0.1639  3.795903
 1176       NaN  2.4727  0.1738  3.786439
 1177       NaN  2.5265  0.1747  3.813609
 1178       NaN  2.4992  0.1765  3.723357
 1179  1.783920  2.4869  0.1765  3.706401
 1180  1.722414  2.5298  0.1772  3.587773
 1181  1.701131     NaN  0.1701  3.589332
 1182  1.698956     NaN  0.1760  3.642491
 1183  1.765315     NaN  0.1715  3.694317
 1184  1.769975     NaN  0.1683  3.793243
 1185  1.796977  2.4993  0.1504  3.866460
 1186  1.787030  2.4953  0.1464  3.920207
 1187  1.781640  2.5650  0.1395       NaN
 1188  1.773772  2.5043  0.1498       NaN
 1189  1.723615  2.5082  0.1579       NaN
 1190  1.710900  2.5386  0.1686       NaN
 1191  1.730096  2.5394  0.1721  3.807987
 1192  1.737664  2.5182  0.1791  3.826752
 1193  1.784289  2.4884     NaN  3.818281
 1194  1.777335  2.5044     NaN  3.824605)
```

We start of with importing the libraries we need for this task and then import the data file. Now that we have the data file we take a quick look at it and discover that the file has many missing values. We also see that all the values are of type float so any converting will not be needed.

In [16]:
```
df.replace(0, np.nan, inplace=True)
df.replace('', np.nan,inplace=True)
df.info()
df.head(), df.tail(20)

df.dropna(inplace=True) #Removes/drops all Nan values in df
miss_data= df.isnull().sum() #checks if there is any missing values
print(miss_data)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1195 entries, 0 to 1194
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Vs      1187 non-null   float64
 1   DEN     1185 non-null   float64
 2   NEU     1184 non-null   float64
 3   Vp      1182 non-null   float64
dtypes: float64(4)
memory usage: 37.5 KB
Vs     0
DEN    0
NEU    0
Vp     0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
Index: 1164 entries, 0 to 1192
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Vs      1164 non-null   float64
 1   DEN     1164 non-null   float64
 2   NEU     1164 non-null   float64
 3   Vp      1164 non-null   float64
dtypes: float64(4)
memory usage: 45.5 KB
```

The method we have used to remove missing values is to replace missing or empty values with NaN and then removing all NaN values. As you can see we now have 0 missing values. We move on to duplicates.

In [17]:
```
dups= df.duplicated()
print(dups.any())
print(df[dups])
```

True

	Vs	DEN	NEU	Vp
1106	1.056314	2.3807	0.3371	2.588172
1107	1.053382	2.3766	0.3452	2.619560
1108	1.066399	2.3660	0.3426	2.626140
1109	1.064992	2.3538	0.3670	2.613025
1110	1.046916	2.3505	0.3509	2.597221
1111	1.033467	2.3533	0.3523	2.581377
1112	1.046970	2.3535	0.3317	2.590031
1113	1.029629	2.3574	0.3455	2.630902
1124	1.072748	2.3358	0.3216	2.584707
1129	1.211407	2.3670	0.3000	2.766731
1132	1.106032	2.3645	0.3233	2.633402
1133	1.072748	2.3358	0.3216	2.584707
1134	1.066803	2.3268	0.3452	2.571996
1135	1.066234	2.3441	0.3472	2.575603
1136	1.056314	2.3807	0.3371	2.588172
1137	1.053382	2.3766	0.3452	2.619560
1138	1.066399	2.3660	0.3426	2.626140
1139	1.064992	2.3538	0.3670	2.613025
1140	1.932877	2.5244	0.1774	3.839216
1141	1.896222	2.5230	0.1629	3.835414
1142	1.765182	2.5164	0.1566	3.728062
1143	1.769825	2.5001	0.1582	3.707501
1144	1.757684	2.5273	0.1603	3.722712
1145	1.797556	2.5380	0.1650	3.742100
1146	1.863159	2.5215	0.1673	3.811086
1147	1.943133	2.5182	0.1637	3.869523
1148	1.956258	2.5212	0.1609	3.919355
1149	2.056708	2.4895	0.1628	3.955760
1150	2.085726	2.4835	0.1612	3.990563
1151	2.066008	2.5158	0.1602	3.995077
1152	2.041241	2.5330	0.1561	4.008643
1153	2.035357	2.5328	0.1554	4.010109
1154	1.956769	2.5552	0.1557	3.991336
1155	1.966766	2.5148	0.1654	3.943375
1156	1.955981	2.5137	0.1667	3.865293
1157	1.912280	2.5407	0.1679	3.845460
1158	1.932862	2.5173	0.1654	3.837336
1159	1.888228	2.5327	0.1548	3.882956
1160	1.877343	2.5392	0.1479	3.865822
1161	1.861162	2.5688	0.1372	3.835718
1162	1.819632	2.5525	0.1317	3.816999
1163	1.821284	2.5467	0.1332	3.810657
1164	1.798054	2.5532	0.1442	3.783755
1165	1.823948	2.5043	0.1615	3.811005
1166	1.835857	2.4815	0.1743	3.780465
1167	1.868822	2.5001	0.1835	3.776137
1168	1.883659	2.4615	0.1809	3.775632
1169	1.892313	2.4193	0.1921	3.770448
1170	1.923482	2.4631	0.1964	3.765808
1171	1.907508	2.4433	0.1975	3.763428
1174	1.834782	2.5383	0.1689	3.782694
1175	1.830254	2.4994	0.1639	3.795903
1179	1.783920	2.4869	0.1765	3.706401
1180	1.722414	2.5298	0.1772	3.587773
1185	1.796977	2.4993	0.1504	3.866460
1186	1.787030	2.4953	0.1464	3.920207
1191	1.730096	2.5394	0.1721	3.807987
1192	1.737664	2.5182	0.1791	3.826752

We check if the data frame contains any duplicates, it does as the dups.any() function returns true. We then print set duplicates and discover quite allot of values. Now starts the process of removing them.

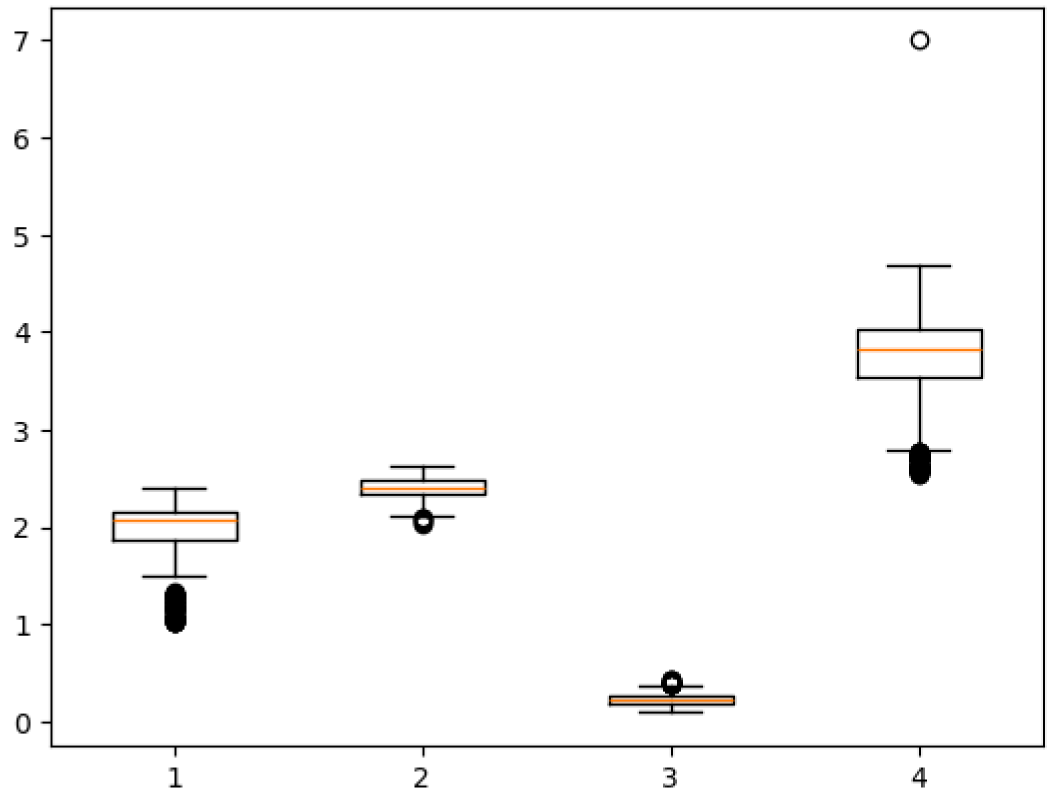
In [18]:
```
df.drop_duplicates(inplace = True)

dups2= df.duplicated()
print(dups2.any())

df.info()
```

```
False
<class 'pandas.core.frame.DataFrame'>
Index: 1106 entries, 0 to 1131
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Vs      1106 non-null   float64
 1   DEN     1106 non-null   float64
 2   NEU     1106 non-null   float64
 3   Vp      1106 non-null   float64
dtypes: float64(4)
memory usage: 43.2 KB
```

After removing the duplicates we again check if there are any and this time it returns false, we have removed all duplicates. We move on to outliers.

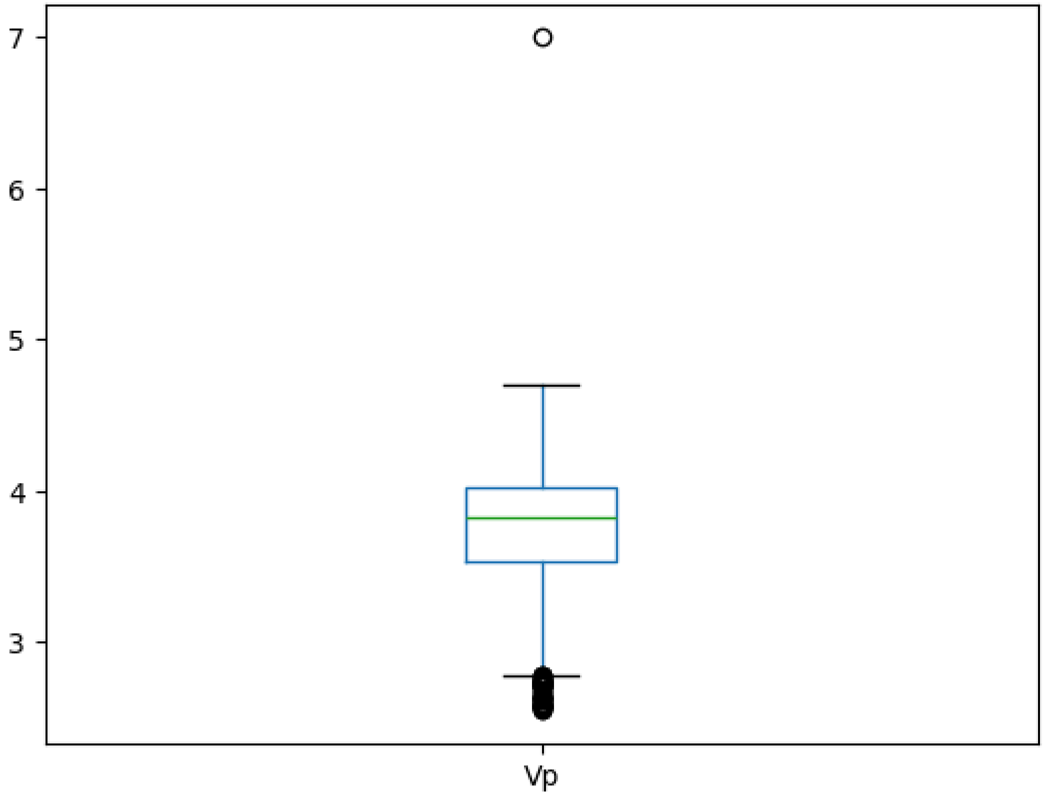
In [19]:
```
df.shape
plt.boxplot(df,widths=0.5)
```

Out[19]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x20932f3e8b0>,
  <matplotlib.lines.Line2D at 0x20932f3e3a0>,
  <matplotlib.lines.Line2D at 0x20932f457f0>,
  <matplotlib.lines.Line2D at 0x20932f45040>,
  <matplotlib.lines.Line2D at 0x209331f2a00>,
  <matplotlib.lines.Line2D at 0x20932b68310>,
  <matplotlib.lines.Line2D at 0x20932f5c130>,
  <matplotlib.lines.Line2D at 0x20932f5c460>],
 'caps': [<matplotlib.lines.Line2D at 0x20932f3ed60>,
  <matplotlib.lines.Line2D at 0x20932f3e160>,
  <matplotlib.lines.Line2D at 0x20932f45130>,
  <matplotlib.lines.Line2D at 0x20932f68970>,
  <matplotlib.lines.Line2D at 0x20932b68070>,
  <matplotlib.lines.Line2D at 0x20932b68be0>,
  <matplotlib.lines.Line2D at 0x20932f5c760>,
  <matplotlib.lines.Line2D at 0x20932f5ce20>],
 'boxes': [<matplotlib.lines.Line2D at 0x20932f3eb20>,
  <matplotlib.lines.Line2D at 0x20932f45c70>,
  <matplotlib.lines.Line2D at 0x20932f68040>,
  <matplotlib.lines.Line2D at 0x20932f5c430>],
 'medians': [<matplotlib.lines.Line2D at 0x20932f45460>,
  <matplotlib.lines.Line2D at 0x20932f68a90>,
  <matplotlib.lines.Line2D at 0x20932b688b0>,
  <matplotlib.lines.Line2D at 0x20932f5c040>],
 'fliers': [<matplotlib.lines.Line2D at 0x20932f45be0>,
  <matplotlib.lines.Line2D at 0x20932f68d30>,
  <matplotlib.lines.Line2D at 0x20932b68640>,
  <matplotlib.lines.Line2D at 0x20932f35dc0>],
 'means': []}
```



By plotting this boxplot we clearly sees that there is many outliers in our data frame. We can also look at a single column.

In [20]:
```
def plot_boxplot(df, ft):
    df.boxplot(column=[ft])
    plt.grid(False)
    plt.show()
plot_boxplot(df, "Vp")
```

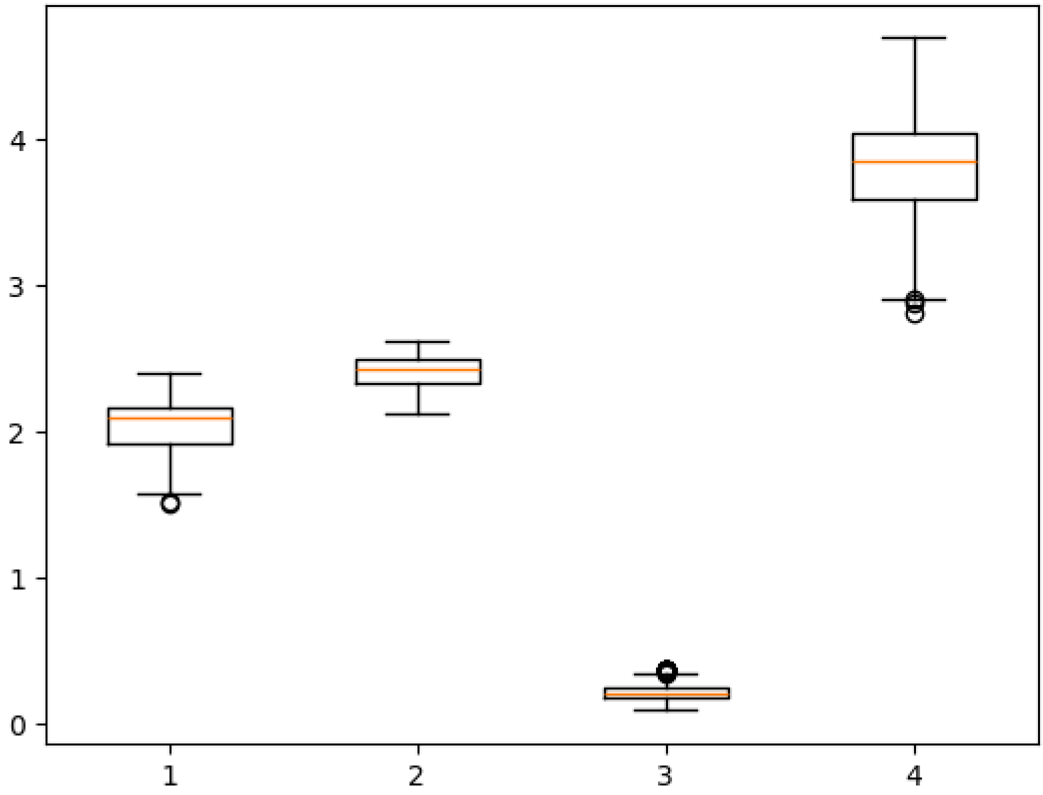


We create a function for plotting a boxplot of a specific column in the data frame. As shown in the boxplot we see many outliers in the Vp column. Now starts the task of removing set outliers.

In [21]:
```
def outliers(df, ft):
    Q1 = df[ft].quantile(0.25)
    Q3 = df[ft].quantile(0.75)
    IQR = Q3 - Q1
    UB = Q3 + 1.5*IQR
    LB = Q1-1.5*IQR
    ls = df.index[ (df[ft] < LB) | (df[ft] > UB)]
    return ls

index_list = []
for f in ['Vs', 'DEN', 'NEU', 'Vp']:
    index_list.extend(outliers(df, f))

def remove(df, ls):
    ls = sorted(set(ls))
    df = df.drop(ls)
    return df

df_cleaned = remove(df, index_list)

print("New shape", df_cleaned.shape)
df_cleaned.info()
```

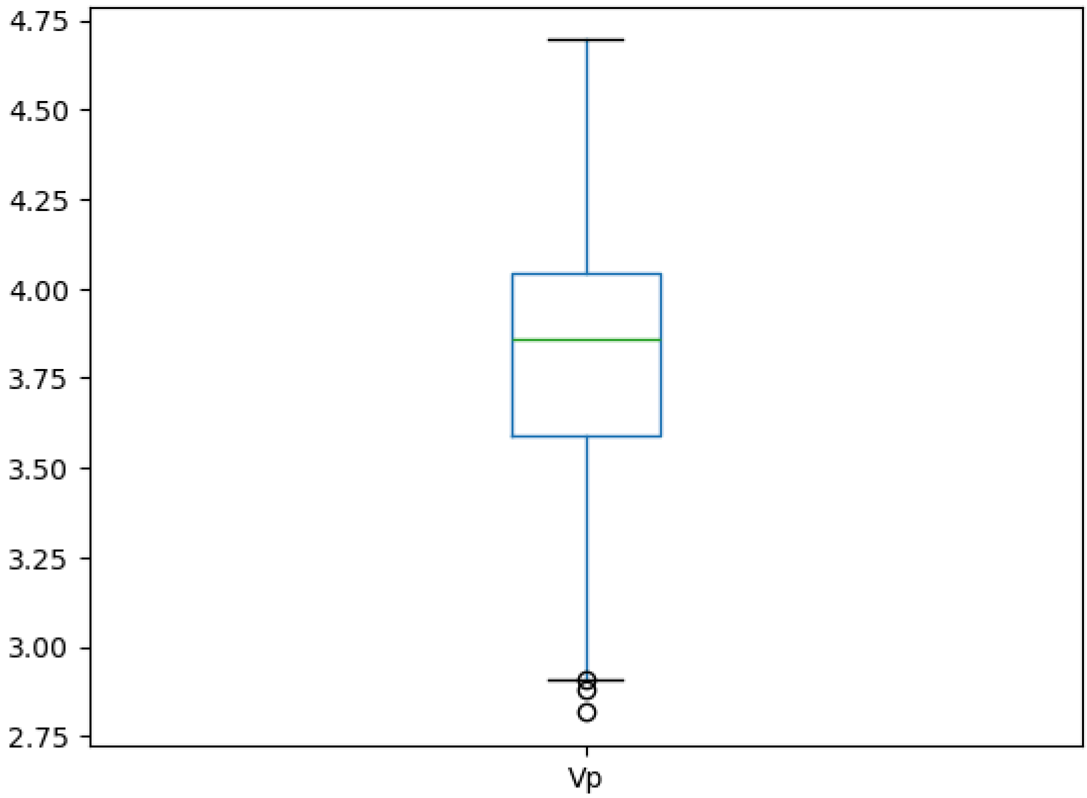
```
plt.boxplot(df_cleaned,widths=0.5)
df_cleaned.to_excel('CleanOutlier1.xlsx', index=False)
```

```
New shape (1024, 4)
<class 'pandas.core.frame.DataFrame'>
Index: 1024 entries, 0 to 1042
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Vs      1024 non-null   float64
 1   DEN     1024 non-null   float64
 2   NEU     1024 non-null   float64
 3   Vp      1024 non-null   float64
dtypes: float64(4)
memory usage: 40.0 KB
```



We create a function for locating all outliers one column at a time using the IQR method. Then we create a for loop that loops for all columns and inserts the outliers for each column into a list. We also create a function that takes a list and a data frame, this function we use to remove all outliers from our data frame using the list of outliers. We now see that almost all of the outliers are removed.

In [22]:
```
plot_boxplot(df_cleaned, "Vp")
```



This is how the Vp column now looks after removing outliers. Much better than before. We will now look at correlation.

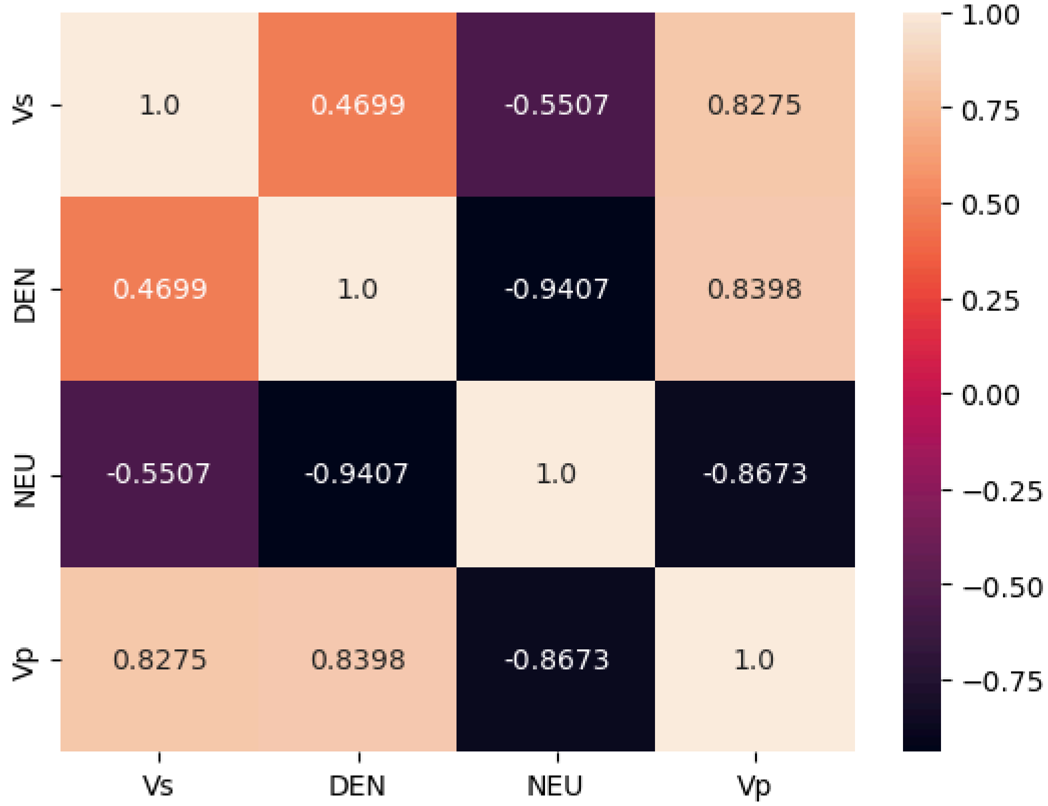
In [23]:

```
df2 = pd.read_excel('CleanOutlier1.xlsx').astype(float)
print(df2.head())
sns.heatmap(df2.corr(), annot = True, fmt= '0.4')
CorrelationData=df2.corr()

CorrelationData
```

```
         Vs     DEN     NEU        Vp
0  1.676857  2.3767  0.2759  3.045533
1  1.677172  2.2101  0.2524  2.974779
2  1.676252  2.1419  0.2591  2.881411
3  1.677030  2.1660  0.2589  2.930981
4  1.684534  2.1193  0.2596  2.910094
```

Out[23]:

	Vs	DEN	NEU	Vp
Vs	1.000000	0.469904	-0.550656	0.827545
DEN	0.469904	1.000000	-0.940704	0.839817
NEU	-0.550656	-0.940704	1.000000	-0.867323
Vp	0.827545	0.839817	-0.867323	1.000000



We display the correlation data in numbers and in a heat map, from this data we see that Vs, DEN and NEU all have a big correlation with Vp. Both Vs and DEN have a positive correlation of almost the exact same size, NEU has a negative correlation which means that when one gets larger the other grows smaller [2]. NEUs correlation is the greatest.

In [24]:
```
CorrelationData['Vp'][abs(CorrelationData['Vp']) > 0.5]
```

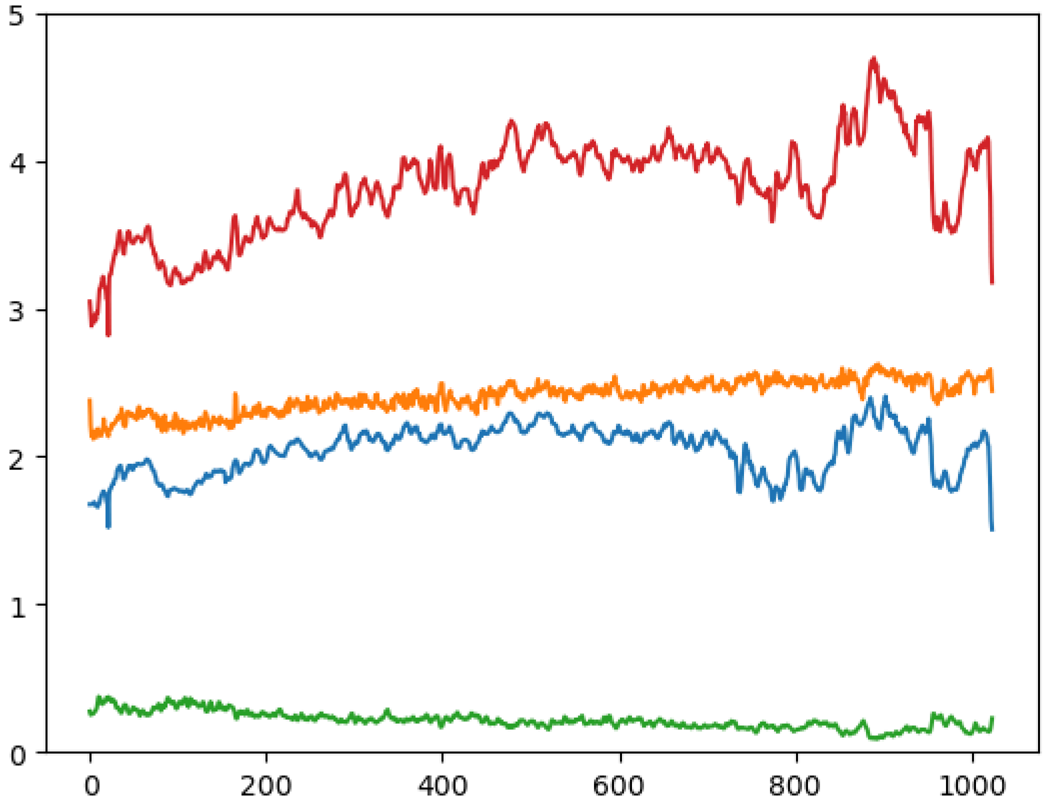
Out[24]:
```
Vs      0.827545
DEN     0.839817
NEU    -0.867323
Vp      1.000000
Name: Vp, dtype: float64
```

According to the project 0.5 is a good threshold to use when selecting features, we see that all our features pass this threshold so we will not drop any features.

In [25]:
```
df3 = pd.read_excel('CleanOutlier1.xlsx')
df3.info()
plt.plot(df3)
plt.ylim(0,5)
```

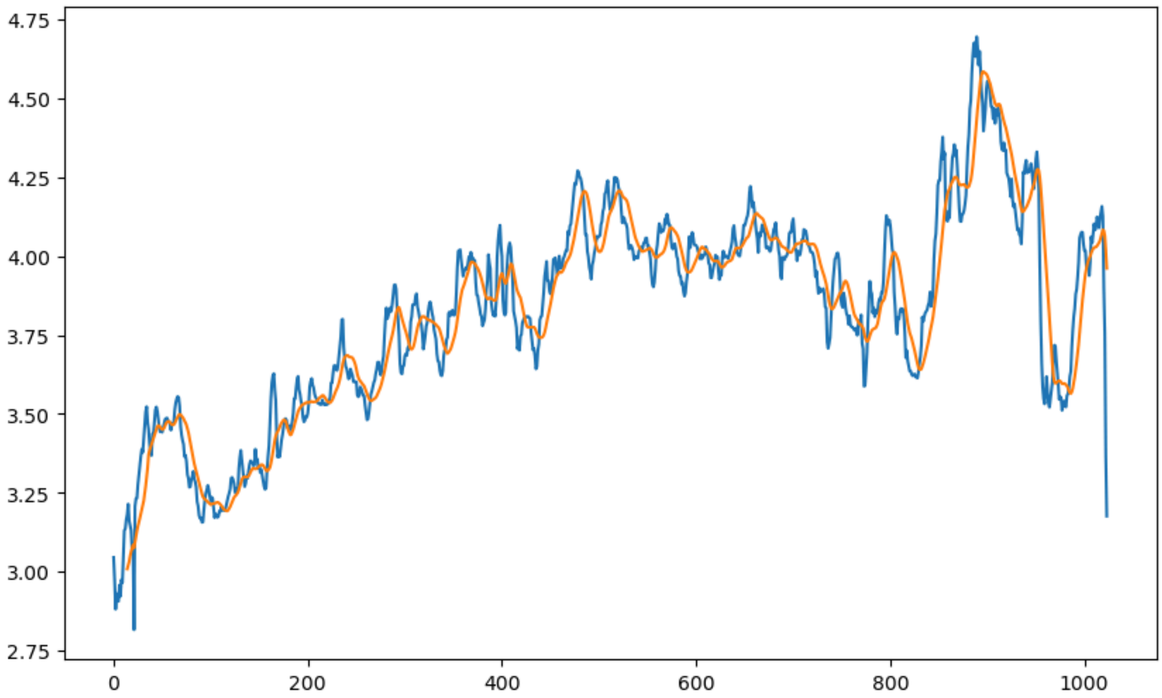
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1024 entries, 0 to 1023
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Vs      1024 non-null   float64
 1   DEN     1024 non-null   float64
 2   NEU     1024 non-null   float64
 3   Vp      1024 non-null   float64
dtypes: float64(4)
memory usage: 32.1 KB
```

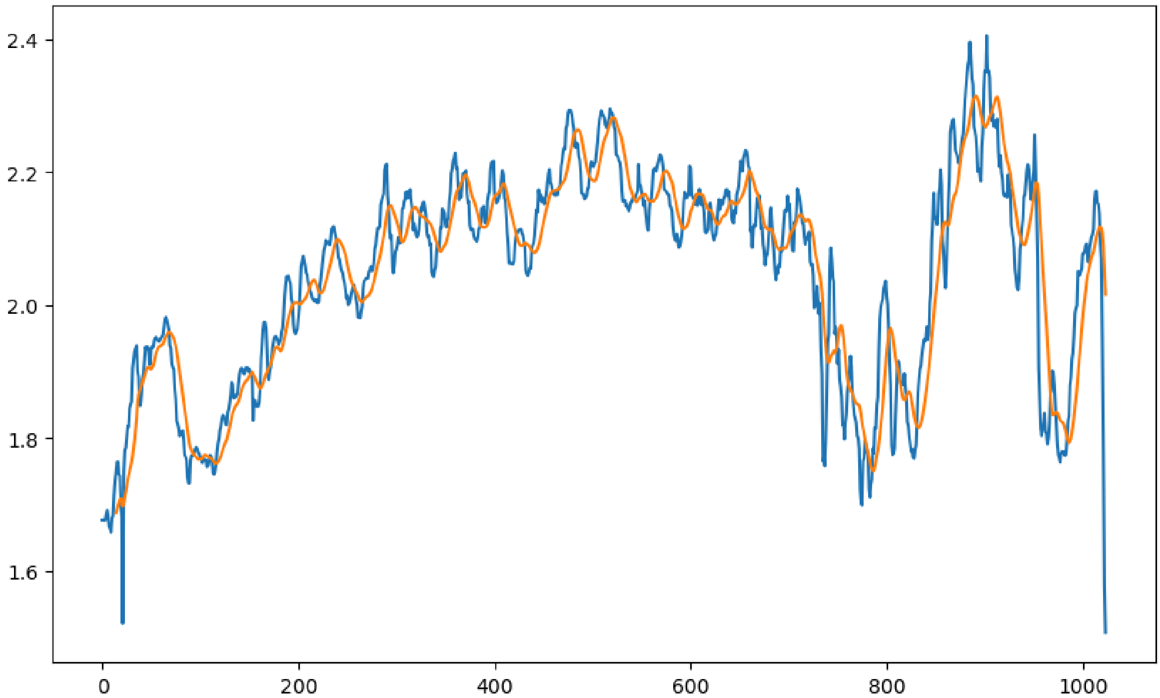
Out[25]: (0.0, 5.0)

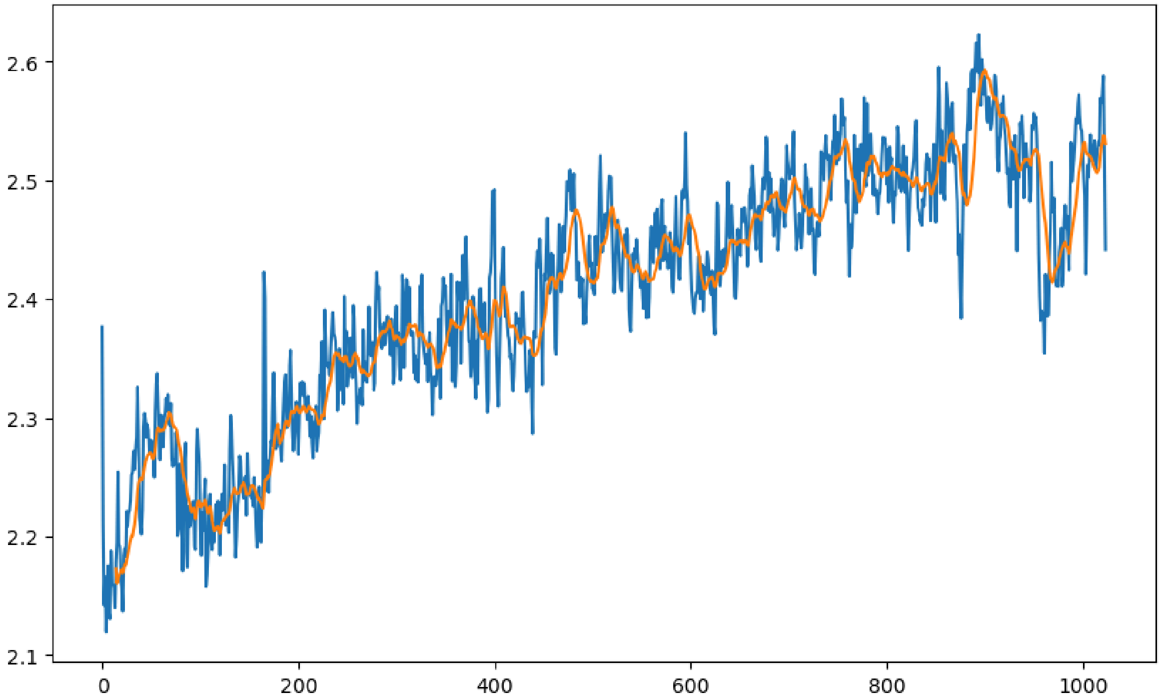


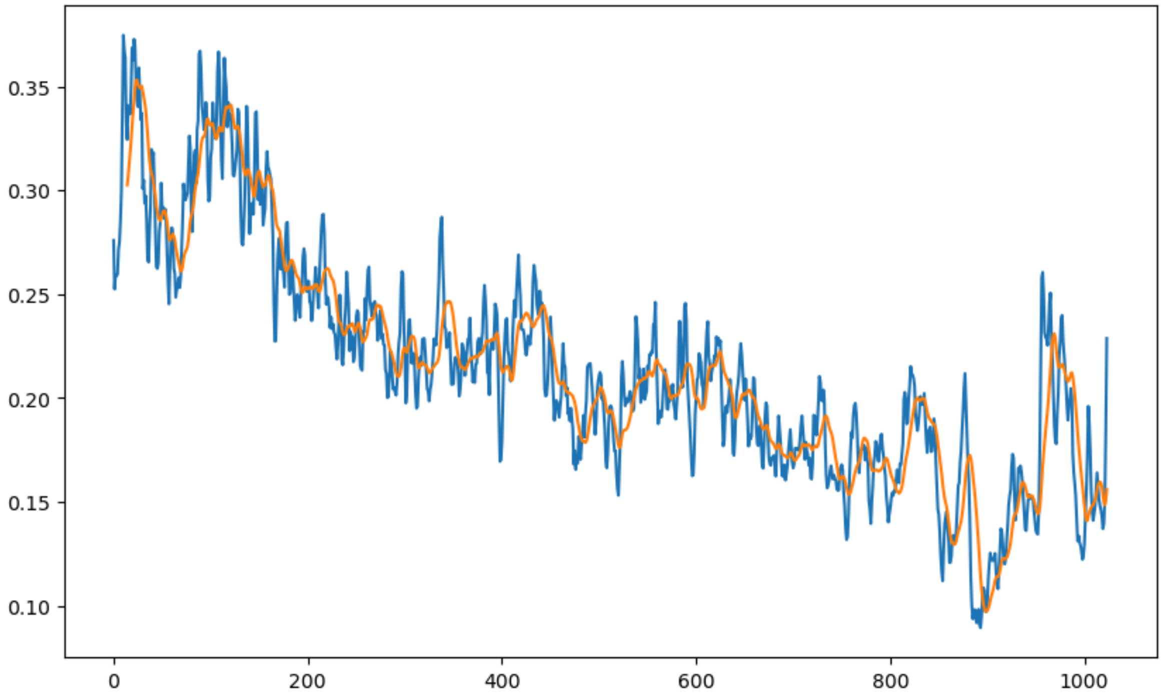
We plot the cleaned data and we see that the data contains some noise, to get clearer data we can apply some filtering/smoothening.

In [26]:
```
plt.plot(df3['Vp'])
dfx1 = df3['Vp'].rolling(window =15).mean().plot(figsize=(10,6))
plt.show()

plt.plot(df3['Vs'])
dfx1 = df3['Vs'].rolling(window =15).mean().plot(figsize=(10,6))
plt.show()

plt.plot(df3['DEN'])
dfx1 = df3['DEN'].rolling(window =15).mean().plot(figsize=(10,6))
plt.show()

plt.plot(df3['NEU'])
dfx1 = df3['NEU'].rolling(window =15).mean().plot(figsize=(10,6))
plt.show()

dataF = df3.rolling(window =15).mean()
dataF.dropna(inplace=True) #Some values went missing after filtering
```


4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
0
200
400
600
800
1000

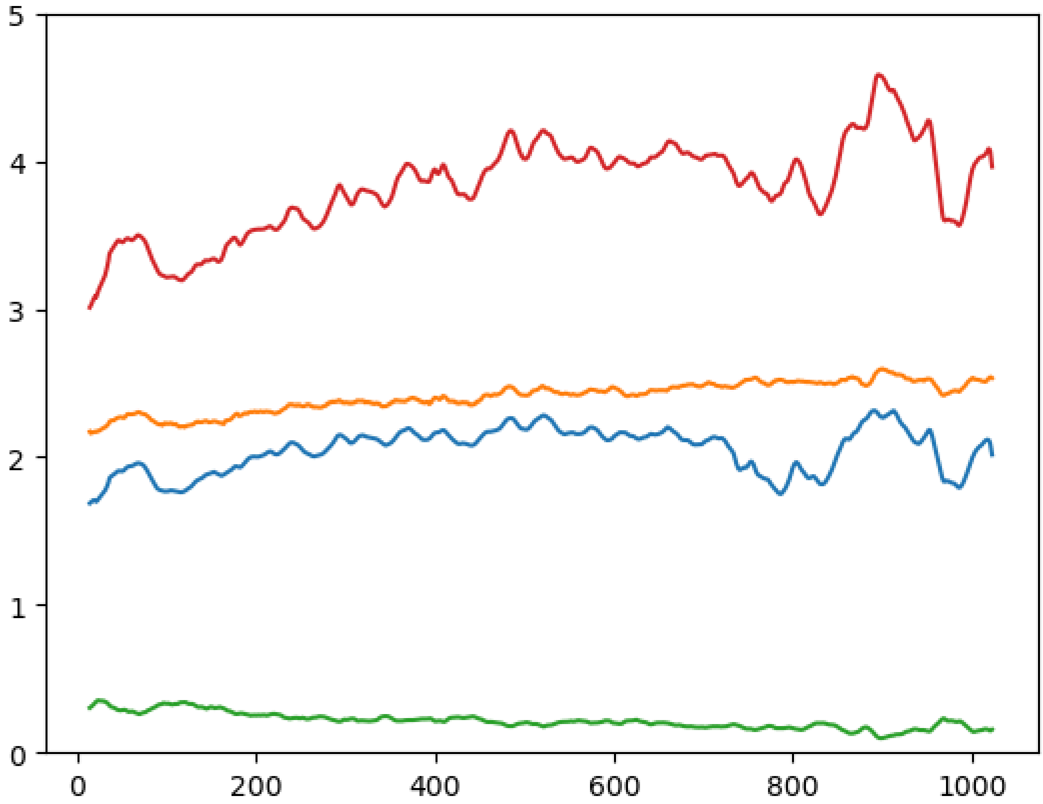

2.4
2.2
2.0
1.8
1.6
0
200
400
600
800
1000





The graphs show the data with and without the filter, orange being with filter. We see that there is less noise/spikes and the data is more consistent.

In [27]:
```
plt.plot(dataF)
plt.ylim(0,5)
plt.show()
dataF.to_excel('CleanedFeatureSelectedFiltered.xlsx', index=False)
```



Again the graph with all the data now shows much less spikes/noise.

Conclusion

The data preprocessing and feature selection steps were key in enhancing the data quality and model readiness. We also saw how big a difference filtering made in terms of noise and spikes. By handling data inconsistencies and focusing on high-correlation features we have improved the accuracy potential of our Vp estimation models. This shows how important data preprocessing is to create a reliable foundation for the ML models to follow.

Reflection

Reflecting over this data preprocessing task, we got a new understanding of how important data integrity is for ML projects. We also learned about how correlation works in terms of feature selection with both negative and positive correlation. Another observation is how big a difference filtering makes, just looking at graphs really shows how important filtering can be when reducing noise.

References

[1]:Amit Kumar Tyagi, Ajith Abraham,"2.5.1 Data preprocessing", Data science for Genomics, 2023

[2]:JOVE 1.13 Correlations, 09.11.2024, https://www.jove.com/science-education/11030/correlation-correlation-coefficient-positive-negative-correlation