Exploratory Data Analysis Project: Pre-Harvest Growth Factors in Hydroponic Farming

https://youtu.be/QGgcoZKympI (https://youtu.be/QGgcoZKympI) The Chamwada Report: Hydroponic Farming in Kenya

'Hydroponic farming within an urban-friendly vertical greenhouse is the latest and greatest in agricultural technology. It can make a huge difference in the world's issues if applied on a larger scale. Healthy food should be available to everyone, and it shouldn't hurt the planet to get it to them. Hydroponics is a sustainable solution to the planet's issues and the people's issues.'Source: https://www.edengreen.com/blog-collection/how-hydroponics-can-save-our-food-system#:~:text=Hydroponic%20food%20is%20grown%20free,their%20reliance%20on%20pesticides%20drastica (https://www.edengreen.com/blog-collection/how-hydroponics-can-save-our-food-system#:~:text=Hydroponic%20food%20is%20grown%20free,their%20reliance%20on%20pesticides%20drastica

I have an immediate family member who is a representative in her Architecture Practice for Sustainability and through discussion I became aware of a future where new buildings include Hydroponic Growing as a response to Soil Exhaustion and Food Scarcity – I also saw the linked Youtube video on how Hydroponic Farming is already saving communities so I felt motivated to base my Exploratory Data Analysis Project around growth factors for Hydroponic Farming. 'Our current agricultural system is up to a huge task: by 2050, we will need to increase food production by about 70% in order to meet the caloric needs of a global population of 9.8 billion people—68% of whom are projected to live in urban areas.' Source: https://psci.princeton.edu/tips/2020/11/9/the-future-of-farming-hydroponics (https://psci.princeton.edu/tips/2020/11/9/the-future-of-farming-hydroponics)
Lettuce is one of the best plants to grow hydroponically ; 'The research concluded that via hydroponics, lettuce yields increase 11 times, while at the same time water consumption reduced by 250 times.' Source: https://www.mdpi.com/1660-4601/12/6/6879 (https://www.mdpi.com/1660-4601/12/6/6879) Because Lettuce plant is so responsive and fast growing I chose to find Datasets linked to Growth outcomes for Hydroponically grown Lettuce

Project Aims:

Research Area: Effect of Pre-havest factors on levels of key metabolic markers in hydroponically grown plants

Aims:

1. Determine which pre-harvest factor (ionic strength of nutrient solution, light intensity and type of nutrient solution) affects the most change in metabolic marker levels
2. Determine the pre-harvest factor intervals that result in the highest metabolic marker levels

Objectives:

1. Implement a Kruskal-Wallis test to determine if a change in each pre-harvest factor results in statistically different metabolic marker levels
2. Implement Kendall's Tau to create a correlation matrix; use this matrix to assess the strength and direction of the correlation between each pre-harvest factor and resulting metabolic marker levels
3. Calculate the average metabolic marker level for each interval of pre-harvest factor ; eg calulate the average glucose level at full ionic strength,

Stages of Analytical Process:

Investigation of Ionic Strength Strength on Metabolic Markers
Exploration of Dataset 1:

- Data Acquisition
- Data Cleansing
- Analysis
- Analysis Technique 1: Kruskal-Wallis Test
- Analysis Technique 2: Kendall's Tau Correlation Matrix
- Analysis Technique 3: Random Forest Regressor to assess feature importance
- Exploration of optimal ionic strength level

Investigation of Light Strength on Metabolic Markers
Exploration of Dataset 2:

- 2. Data Cleansing
- 2.-Analysis
- 2. Analysis Technique 1: Kruskal-Wallis Test
- 2. Analysis Technique 2: Kendall's Tau Correlation Matrix
- Exploration of optimal light strength level

Investigation of Effect of Nutrient Solution Type on Metabolic Markers
Exploration of Dataset 3:

- 3. Data Cleansing
- 3.-Analysis
- 3. Analysis Technique 1: Kruskal-Wallis Test
- 3. Analysis Technique 2: Kendall's Tau Correlation Matrix
- Exploration of optimal Nutrient Solution Type

Comparison of Kendall's Tau Results for Each Dataset
Conclusion and Critical Evaluation
Ethical Statement

Appendix:

- Web scraping of Comparison Dataset
- Analysis Technique 3: Linear Regression

Exploration of Dataset 1:

Data Acquisition

Dataset: Dataset on the Effects of Different Pre-Harvest Factors on the Metabolomics Profile of Lettuce (Lactuca sativa L.) Leaves

Authors:

- Department of Agricultural Sciences, University of Naples Federico II, 80055 Portici, Italy
- Department for Sustainable Food Process, Research Centre for Nutrigenomics and Proteomics, University Cattolica del Sacro Cuore, 29122 Piacenza, Italy
- Research Centre for Genomics and Bioinformatics (CREA-GB), via San Protaso 302, 29017 Fiorenzuola d'Arda, Italy

Published: 15 December 2020

Source: https://www.mdpi.com/2306-5729/5/4/119/pdf (https://www.mdpi.com/2306-5729/5/4/119/pdf) (Published Paper) https://www.mdpi.com/2306-5729/5/4/119?type=check_update&version=2 (https://www.mdpi.com/2306-5729/5/4/119?type=check_update&version=2) (Website hosting published paper)

Chicago Reference: Corrado, Giandomenico, Luigi Lucini, Begoña Miras-Moreno, Leilei Zhang, Biancamaria Senizza, Boris Basile, and Youssef Rouphael. 2020. "Dataset on the Effects of Different Pre-Harvest Factors on the Metabolomics Profile of Lettuce (Lactuca sativa L.) Leaves" Data 5, no. 4: 119. https://doi.org/10.3390/data5040119 (https://doi.org/10.3390/data5040119)

In [1]:

```
import pandas as pd
import statsmodels.api as sm
```

In [2]:

```
#dataset was downloaded from link above as excel file
raw_data = pd.read_excel('Dataset Metabolomics.xlsx', sheet_name=0)
```

'The column/sample's cells report the peak intensity value (raw abundance of extracted ion current)'

In [3]:

```
#convert the initial read of data into a pandas dataframe ;
#I chose to use pandas dataframes as Pandas can efficiently import and handle large dataset
#Source: https://data-flair.training/blogs/advantages-of-python-pandas/
#Pandas also provides key utilities for formatting, filtering and representing data - helpi
#during analysis

raw_data_df = pd.DataFrame(raw_data)
raw_data_df
```

Out[3]:

	Genotype	GS	GS.1	GS.2	GS.3	GS.4	GS.5	GS.6	
0	Ionic strenght	FS	FS	FS	FS	FS	FS	HS	
1	CPDs ID	1-r001	1-r002	2-r001	2-r002	3-r001	3-r002	4-r001	
2	CPD-4887	2812738	2696882	2317435	2210526	3007915	2993476	2549627	2
3	CPD-8797	188980	198695	257806	263646	2045953	2051394	1	
4	CPD-10759	326603	915013	721808	713565	1140404	1	566768	
...	...	...	...	...	...	...	...	...	
3974	CPD-5661	3515827	3016040	3341571	1	1209546	1206648	2744050	3
3975	CPD-11421	1232501	1	708735	710682	532125	1	524114	
3976	ZOLEDRONATE	228202	202170	226688	208382	1	248815	1	
3977	ZYMOSTEROL	1527896	1	2586168	2625316	6462030	6357935	4624190	4
3978	CPD-4581	5835241	5837042	16165811	16140914	31485504	1	22247708	22

3979 rows × 39 columns

Data Cleansing

Clean Format of Data

Define a function to format each dataset:

In [4]:

```
import numpy as np
def formatData(raw_data_df):
    #this dataset uses 1 to represent missing values - convert this to NaN ready for data a
    nan_data = raw_data_df.replace(1, np.nan)
    #transpose data so that metabolic markers are the column names; for ease of reference d
    nan_data = nan_data.transpose()
    #remove composite spectrum data - this is not required at the moment
    #the composite spectrum provides further identification of the chemical compound being
    nan_data.drop(nan_data.tail(1).index,inplace=True)

    #row index system is currently genotype
    #insert column for genotype - so that this variable can be included in data analysis
    indexNamesArr = nan_data.index.values
    nan_data.insert(loc=0, column='Genotype', value=indexNamesArr)
    #reset row index to be number system
    nan_data = nan_data.reset_index(drop=True)

    #column names are currently in first row of dataframe - correct this
    nan_data.columns = nan_data.iloc[0].to_list()
    nan_data.drop(index=nan_data.index[0],
            axis=0,
            inplace=True)
    #swap chemical CPD codes for compound names to make the dataset more easily interpretab
    new_col_names = nan_data.columns.values[:3]
    chem_names =  nan_data.iloc[-1].tolist()
    chem_names = chem_names[3:]
    new_col_names = np.concatenate((new_col_names, chem_names))
    nan_data.columns = new_col_names
    #remove last row of data as chemical names are now contained in the column headers
    #remove duplicate data
    nan_data.drop(nan_data.tail(1).index, inplace=True)

    #read_excel has interpret genotype column as GS.1, GS.2, GS.3... (rather than categotic
    corrected_genotypes = []

    for genotype in list(nan_data['Genotype']):
        if 'G' in genotype:
            corrected_genotypes.append('GS')
        else:
            corrected_genotypes.append('RS')

    nan_data['Genotype'] = corrected_genotypes
    return nan_data
```

In [5]:

```
ionic_strength_df = formatData(raw_data_df)
ionic_strength_df.head()
```

Out[5]:

	Genotype	Ionic strenght	CPDs ID	(-)-(4S)-α-terpineol	(-)-α-amorphene	(-)-α-bisabolol	(-)-α-cuprenene	(-)-α-pinene	(- caryophyll
1	GS	FS	1-r001	2812738	188980	326603	188980	1927143	188
2	GS	FS	1-r002	2696882	198695	915013	198695	1820475	198
3	GS	FS	2-r001	2317435	257806	721808	257806	1594339	257
4	GS	FS	2-r002	2210526	263646	713565	263646	1508726	263
5	GS	FS	3-r001	3007915	2045953	1140404	2045953	1968036	2045

5 rows × 3980 columns

Explore level of missing data

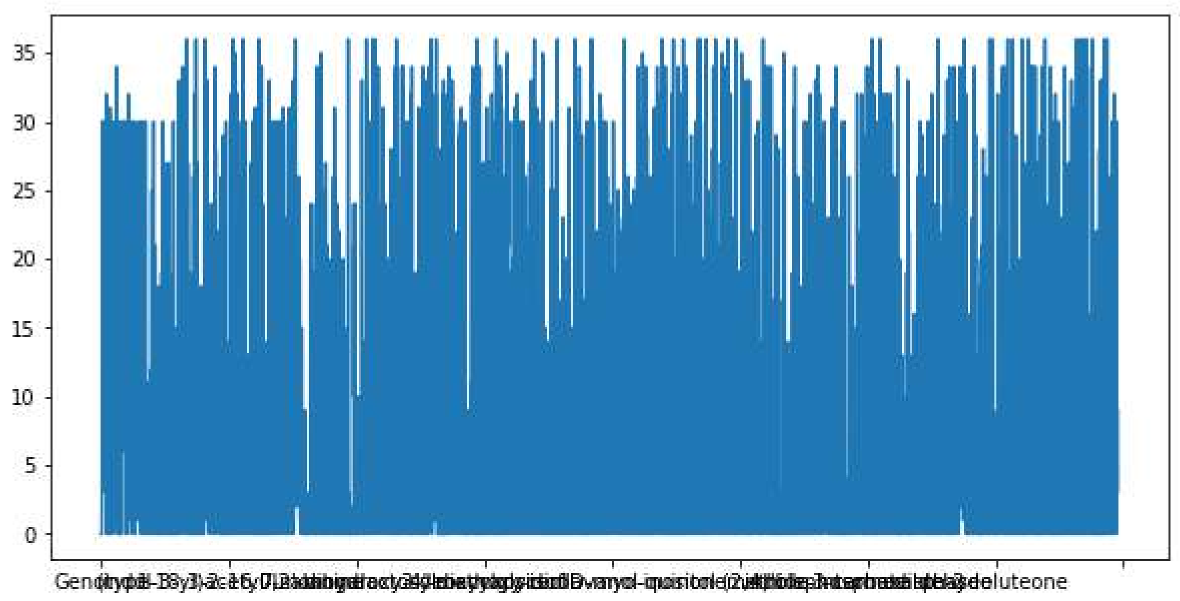
According to the following source, 'The proportion of missing data is directly related to the quality of statistical inferences' Source: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3701793/ (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3701793/) in light of this information I will assess the level of missing data in each dataset:

In [6]:

```
#visualise the missing data
missing = ionic_strength_df.isnull().sum()
missing.plot(figsize=(10,5))
```

Out[6]:

```
<AxesSubplot:>
```



Calculate number of compounds missing more than 95% of data:

In [7]:

```
#calculation sourced from
#https://www.thiscodeworks.com/python-find-out-the-percentage-of-missing-values-in-each-col
percent_missing = ionic_strength_df.isnull().sum() * 100 / len(ionic_strength_df)
#the following compounds are missing more than 95% of the data
count = 0
for index, value in percent_missing.items():
    if(value > 95):
        count +=1
#create variable storing total number of compounds in dataset
#first 3 rows are genotype, ionic strength and CPD ID - these need to removed / sliced from
num_compounds = len(list(ionic_strength_df)[3:])
percent_variables_missing = (count/num_compounds) *100
print(percent_variables_missing)
```

```
1.5086748805632386
```

Dataset evaluation: The cell above identifies that some compounds have more than 95% of data missing - this may be indicative of potential flaws in data collection - however the number of variables with more than 95% of missing data is estimated that 1.509%.

Identifying potential features (level of missing data less than 10%)

"Statistical guidance articles have stated that bias is likely in analyses with more than 10% missingness and that if more than 40% data are missing in important variables then results should only be considered as hypothesis generating [18], [19]."
Source: [https://www.sciencedirect.com/science/article/pii/S0895435618308710 (https://www.sciencedirect.com/science/article/pii/S0895435618308710)](https://www.sciencedirect.com/science/article/pii/S0895435618308710)

Define a function that will remove compounds with more than 10% missingingness: this function will be applied to each dataset:

In [8]:

```
def removeBiasVar(data_df):
    #calculation sourced from
    #https://www.thiscodeworks.com/python-find-out-the-percentage-of-missing-values-in-each
    percent_missing = data_df.isnull().sum() * 100 / len(data_df)

    #create an array of the names of chemical compounds that have more than 10% missing dat
    non_bias_data = []

    #method informed by
    #https://www.codegrepper.com/code-examples/python/iterate+over+pandas+series
    for index, value in percent_missing.items():
        if(value <= 10):
            non_bias_data.append(index)

    #evaluation of this approach
    num_compounds = len(list(data_df)[3:])
    print(f"Remaining % of Data: {(len(non_bias_data)/num_compounds)*100}")
    #remove potentially biased (missing more than 10% of data) from dataframe
    #create a copy for version control
    non_bias_df = data_df.copy()

    to_remove = []
    for (columnName, columnData) in non_bias_df.iteritems():
        if columnName not in non_bias_data:
            to_remove.append(columnName)

    for compound in to_remove:
        #allow for duplicate data
        if f"{compound}" in non_bias_df:
            non_bias_df.drop(compound, axis=1,inplace=True)

    return non_bias_df
```

In [9]:

```
non_bias_df = removeBiasVar(ionic_strength_df)
```

```
Remaining % of Data: 53.557958259994976
```

Evaluation of this approach:

While this approach reduces the level of bias in the dataset - it is removing approximately 47% of the data - the current dataset still contains the following number of compounds:

In [10]:

```
len(non_bias_df.columns[3:])
```

Out[10]:

```
2131
```

Prepare Data for Outlier Detection and Iterative Imputer

Sklean Iterative imputer requires dummy values, otherwise the following error is returned: ValueError: could not convert string to float: 'GS'

In [11]:

```
#map ionic strength types to numbers so that rows can be grouped using sort_values (ready f
#outlier detection is being done within each ionic strength band (FS,HS,QS)
non_bias_dummy_df = non_bias_df.copy()
non_bias_dummy_df['Genotype'] = non_bias_dummy_df['Genotype'].map({'GS': 1, 'RS':0})
non_bias_dummy_df['Ionic strenght'] = non_bias_dummy_df['Ionic strenght'].map({'FS': 1.5, '
```

In [12]:

```
#remove any duplicate columns names
#method informed by
#https://stackoverflow.com/questions/14984119/python-pandas-remove-duplicate-columns
non_bias_dummy_df = non_bias_dummy_df.loc[:,~non_bias_dummy_df.columns.duplicated()].copy()
non_bias_dummy_df.head()
```

Out[12]:

	Genotype	Ionic strenght	CPDs ID	(-)-(4S)-α-terpineol	(-)-endo-fenchol	(-)-5'-demethylyatein	(-)-9β-pimara-7,15-dien-19-ol	(-)-9β-pimara-7,15-diene	(-)-borneol
1	1	1.5	1-r001	2812738	2812738	728482	656928	1791541	2812738
2	1	1.5	1-r002	2696882	2696882	642408	629786	1812067	2696882
3	1	1.5	2-r001	2317435	2317435	1432100	647746	1454032	2317435
4	1	1.5	2-r002	2210526	2210526	2420230	643771	1477626	2210526
5	1	1.5	3-r001	3007915	3007915	1081016	727991	2018411	3007915

5 rows × 2125 columns

Detecting Potential Outliers using the Interquartile

Range Method

I shall calculate the lower and upper limits using the following formula:
Lower limit = Q1 – 1.5*IQR (Q1 = 25th percentile)*
*Upper limit = Q3 + 1.5*IQR (Q3 = 75th percentile)

In [13]:

```
#create a copy for version control - this will allow for comparisons of the dataset before
outliers_df = non_bias_dummy_df.copy()
```

Group Ionic strength rows together: for calculation of outliers within interval (FS,HS,QS)

In [14]:

```
sorted_data = outliers_df.sort_values('Ionic strenght', inplace=False)
#use the drop paramter to prevent old index system being added as a column
sorted_data.reset_index(drop=True, inplace=True)
sorted_data
```

Out[14]:

	Genotype	Ionic strenght	CPDs ID	(-)-(4S)-α-terpineol	(-)-endo-fenchol	(-)-5'-demethylyatein	(-)-9β-pimara-7,15-dien-19-ol	(-)-9β-pimara-7,15-diene	(-)-borneol	burse
0	1	0.50	9-r002	2525843	2525843	1067590	4247983	1526010	2525843	133
1	0	0.50	Lettuce_17-r002	1433160	1433160	1035707	18603212	458057	1433160	239
2	0	0.50	Lettuce_17-r001	NaN	NaN	1093665	18237296	462266	NaN	1
3	0	0.50	Lettuce_16-r002	NaN	NaN	1128519	597043	2152103	NaN	18
4	0	0.50	Lettuce_16-r001	1468904	1468904	1193380	598109	2165609	1468904	14

Define a Function to remove outliers from columns based on list of columns; this can be applied to each dataset

In [15]:

```
import numpy as np
#largest element must come first
#eg categoricalValues = [1.5, 0.75, 0.5]
def removeOutliers(colNames, dataframe, independentVarName, categoricalValues):
    return_df = dataframe.copy()

    for colName in colNames:
        #categorical variable with 3 values
        if (len(categoricalValues) ==3):
            #*********************************************************************
            #detect and remove outliers from ionic strength FS values
            full_ion_stren_df = dataframe.loc[dataframe[independentVarName] == categoricalV
            full_ion_stren_values = np.array(full_ion_stren_df[colName])
            #the following line of code prevents the following erro:
            #TypeError: ufunc 'isnan' not supported for the input types,
            #and the inputs could not be safely coerced to any supported types according to
            full_ion_stren_values = [i for i in full_ion_stren_values]

            #method from https://machinelearningmastery.com/how-to-use-statistics-to-identi
            q25, q75 = np.nanpercentile(full_ion_stren_values, 25), np.nanpercentile(full_i
            #calculate outlier cut-off
            iqr = q75 - q25
            cut_off = iqr * 1.5
            lower, upper = q25 - cut_off, q75 + cut_off

            # remove outliers
            outliers_removed_1 = [x if (x > lower and x < upper) else np.nan for x in full_

            #*********************************************************************
            #detect and remove outliers from ionic strength HS values
            half_ion_stren_df = dataframe.loc[dataframe[independentVarName] == categoricalV
            half_ion_stren_values = np.array(half_ion_stren_df[colName])
            half_ion_stren_values = [i for i in half_ion_stren_values]

            #method from https://machinelearningmastery.com/how-to-use-statistics-to-identi
            q25, q75 = np.nanpercentile(half_ion_stren_values, 25), np.nanpercentile(half_i
            q25 = np.nanpercentile(half_ion_stren_values, 25)
            #calculate outlier cut-off
            iqr = q75 - q25
            cut_off = iqr * 1.5
            lower, upper = q25 - cut_off, q75 + cut_off

            # remove outliers
            outliers_removed_2 = [x if x > lower and x < upper else np.nan for x in half_io

            #*********************************************************************
            #detect and remove outliers from ionic strength QS values
            quarter_ion_stren_df = dataframe.loc[dataframe[independentVarName] == categoric
            quarter_ion_stren_values = np.array(quarter_ion_stren_df[colName])
            quarter_ion_stren_values = [i for i in quarter_ion_stren_values]

            #method from https://machinelearningmastery.com/how-to-use-statistics-to-identi
            q25, q75 = np.nanpercentile(quarter_ion_stren_values, 25), np.nanpercentile(qua
            #calculate outlier cut-off
            iqr = q75 - q25
            cut_off = iqr * 1.5
            lower, upper = q25 - cut_off, q75 + cut_off

            # remove outliers
            outliers_removed_3 = [x if x > lower and x < upper else np.nan for x in quarter


            #concatenate new values for column and replace
            all_outliers_removed = outliers_removed_3 + outliers_removed_2 + outliers_remov

            return_df[colName] = all_outliers_removed

        #dichotomous categorical variable
        elif (len(categoricalValues) ==2):
            #***********************************************************************
            #detect and remove outliers from ionic strength FS values
            full_ion_stren_df = dataframe.loc[dataframe[independentVarName] == categoricalV
            full_ion_stren_values = np.array(full_ion_stren_df[colName])
            #the following line of code prevents the following erro:
            #TypeError: ufunc 'isnan' not supported for the input types,
            #and the inputs could not be safely coerced to any supported types according to
            full_ion_stren_values = [i for i in full_ion_stren_values]

            #method from https://machinelearningmastery.com/how-to-use-statistics-to-identi
            q25, q75 = np.nanpercentile(full_ion_stren_values, 25), np.nanpercentile(full_i
            #calculate outlier cut-off
            iqr = q75 - q25
            cut_off = iqr * 1.5
            lower, upper = q25 - cut_off, q75 + cut_off

            # remove outliers
            outliers_removed_1 = [x if (x > lower and x < upper) else np.nan for x in full_

            #***********************************************************************
            #detect and remove outliers from ionic strength HS values
            half_ion_stren_df = dataframe.loc[dataframe[independentVarName] == categoricalV
            half_ion_stren_values = np.array(half_ion_stren_df[colName])
            half_ion_stren_values = [i for i in half_ion_stren_values]

            #method from https://machinelearningmastery.com/how-to-use-statistics-to-identi
            q25, q75 = np.nanpercentile(half_ion_stren_values, 25), np.nanpercentile(half_i
            q25 = np.nanpercentile(half_ion_stren_values, 25)
            #calculate outlier cut-off
            iqr = q75 - q25
            cut_off = iqr * 1.5
            lower, upper = q25 - cut_off, q75 + cut_off

            # remove outliers
            outliers_removed_2 = [x if x > lower and x < upper else np.nan for x in half_io

            #concatenate new values for column and replace
            all_outliers_removed = outliers_removed_2 + outliers_removed_1

            return_df[colName] = all_outliers_removed

    return return_df
```

In [16]:

```
no_outliers_df = removeOutliers(sorted_data.columns[3:], sorted_data, 'Ionic strenght' ,[1.
```

In [17]:

```
#confirm that outliers have been removed / the dataset has been modified
sorted_data.equals(no_outliers_df)
```

Out[17]:

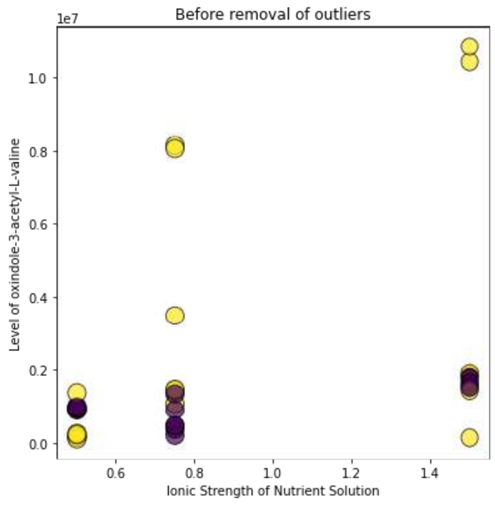
```
False
```

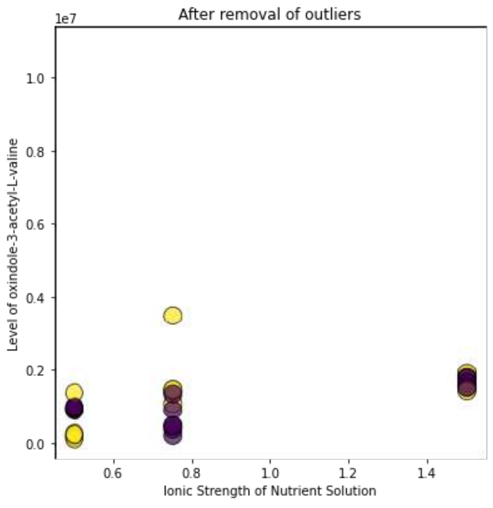
Assess removal of outliers through visualisation of sample data

In [18]:

```python
# #method to color coordinate genotype informed by:
# #https://stackoverflow.com/questions/59232073/scatter-plot-with-3-variables-in-matplotlib
import matplotlib.pyplot as plt
import numpy as np


fig = plt.figure()
fig = plt.figure(figsize=(15, 6))
ax1 = fig.add_subplot(1,2,2)

x = no_outliers_df['Ionic strenght']
y = no_outliers_df['oxindole-3-acetyl-L-valine']
z = no_outliers_df['Genotype']

ax1.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

ax1.set_xlabel('Ionic Strength of Nutrient Solution')
ax1.set_ylabel(f"Level of oxindole-3-acetyl-L-valine")
ax1.set_title("After removal of outliers")
#add sharex parameter to ensure both axis are on the same scale ; for comparison
ax2 = fig.add_subplot(1,2,1, sharex=ax1, sharey=ax1)

x = sorted_data['Ionic strenght']
y = sorted_data['oxindole-3-acetyl-L-valine']
z = sorted_data['Genotype']

ax2.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

ax2.set_xlabel('Ionic Strength of Nutrient Solution')
ax2.set_ylabel(f"Level of oxindole-3-acetyl-L-valine")
ax2.set_title("Before removal of outliers")

plt.subplots_adjust(left=0.1,
            bottom=0.1,
            right=0.9,
            top=0.9,
            wspace=0.5,
            hspace=0.4)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```





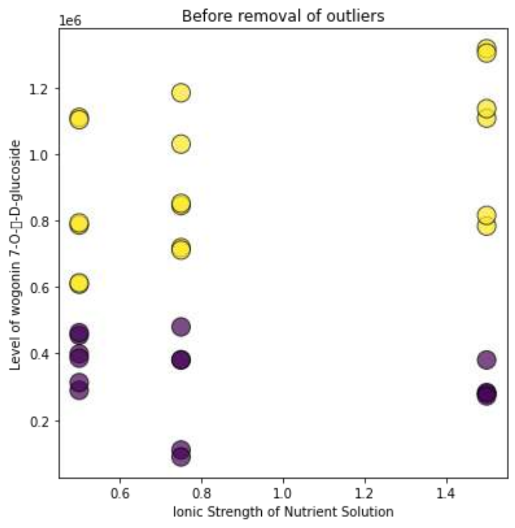
(Color scheme in graphs used to represent lettuce genotype)
Function removeOutliers() seems to have appropriately removed the outliers at ionic strength levels 0.75 and 1.50.

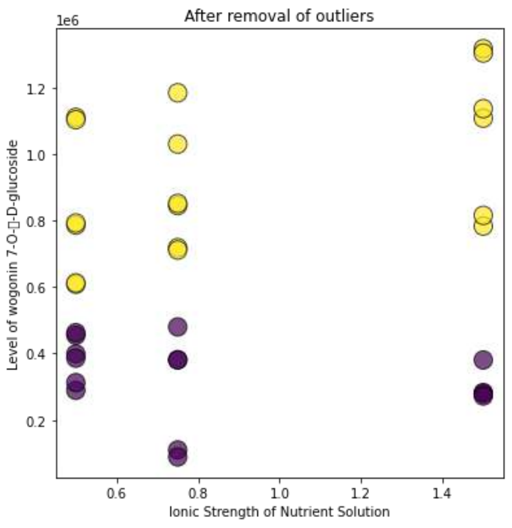
In [19]:

```python
# #method to color coordinate genotype informed by:
# #https://stackoverflow.com/questions/59232073/scatter-plot-with-3-variables-in-matplotlib
import matplotlib.pyplot as plt
import numpy as np


fig = plt.figure()
fig = plt.figure(figsize=(15, 6))


ax1 = fig.add_subplot(1,2,2)

x = no_outliers_df['Ionic strenght']
y = no_outliers_df['wogonin 7-O-𝛽-D-glucoside']
z = no_outliers_df['Genotype']

ax1.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

ax1.set_xlabel('Ionic Strength of Nutrient Solution')
ax1.set_ylabel(f"Level of wogonin 7-O-𝛽-D-glucoside")
ax1.set_title("After removal of outliers")

ax2 = fig.add_subplot(1,2,1, sharex=ax1, sharey=ax1)

x = sorted_data['Ionic strenght']
y = sorted_data['wogonin 7-O-𝛽-D-glucoside']
z = sorted_data['Genotype']

ax2.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

ax2.set_xlabel('Ionic Strength of Nutrient Solution')
ax2.set_ylabel(f"Level of wogonin 7-O-𝛽-D-glucoside")
ax2.set_title("Before removal of outliers")

plt.subplots_adjust(left=0.1,
            bottom=0.1,
            right=0.9,
            top=0.9,
            wspace=0.5,
            hspace=0.4)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>

C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_ag
g.py:240: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_ag
```

```
g.py:203: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0, flags=flags)
```





function removeOutliers() seems to have not removed any data from this sample - this seems appropriate as the graphs are not showing any extreme outliers

Potential impacts of outlier removal:
I removed outliers to help reduce data skewness that may negatively impact statistically analysis - A potential disadvantage of removing outliers is that it may remove data that is biological significant ; an outlier may be caused by an independent varibale that is not measured by these datasets. Pierre Lafaye de Micheaux, Author and Statistician states that the detection of outilers is important as they can lead to new scientific discoverties.

Replacing NaN values using Imputation

I will replace Nan values using imputation; my initial research identified 3 possible imputer methods: SKlearn Simple Imputer, SKlearn KNN Imputer and Sklearn Iterative Imputer. Further research identified that Simple Imputer is a univariate approach which can limit accuracy of imputed values - Sklean KNN method uses a specifed number of neighboring rows to imputed values, this may not be suitable for this particular dataset; for example when imputing the value of the first QS value (see above), the KNN algorithm may use the previous 2 rows, which are data entries for the HS interval - this may lead the KNN algorithm to incorrectly impute a value. Source: Source: https://www.youtube.com/watch?v=m_qKhnaYZlc (https://www.youtube.com/watch?v=m_qKhnaYZlc)
The sklearn iterative imputer builds a regression model; it uses 'A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion.' source: https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html (https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html)
Based on this information, Sklearn Iterative Imputer seems to be the most appropriate imputer to apply to this dataset:

In [20]:

```
import sklearn
print('The scikit-learn version is {}.'.format(sklearn.__version__))
```

```
The scikit-learn version is 0.24.2.
```

In [21]:

```
# explicitly require this experimental feature
from sklearn.experimental import enable_iterative_imputer  # noqa
# now you can import normally from sklearn.impute
from sklearn.impute import IterativeImputer
```

In [22]:

```
# explicitly require this experimental feature
from sklearn.experimental import enable_iterative_imputer  # noqa
# can now import normally from sklearn.impute
from sklearn.impute import IterativeImputer

def imputeValues(list_col_names, dataframe, dependentVar):
    for compound in list_col_names:
        if dataframe[f"{compound}"].isnull().values.any():
            temp_df = dataframe[['Genotype', dependentVar]].copy()
            temp_df[f"{compound}"] = dataframe[f"{compound}"]
            #index starts at 1
            imputer = IterativeImputer()
            temp_df = imputer.fit_transform(temp_df)
            #iterative imputer returns a numpy.ndarray
            #convert numpy.ndarray to pandas dataframe
            #for easier updating of data in table (repalce data with results from iterative
            temp_pandas_df = pd.DataFrame(temp_df, columns=['Genotype', dependentVar, f"{co
            #index starts at 0 - match the index of the original dataframe
            #otherwise last line of data is lost
            #temp_pandas_df.index += 1
            #update the values in the table to include the imputed values
            dataframe[f"{compound}"] = temp_pandas_df[f"{compound}"]
```

In [23]:

```
col_names = list(no_outliers_df)
col_names = col_names[3:]

imputeValues(col_names, no_outliers_df, 'Ionic strenght')
```

In [24]:

```
#confirm that Iterative Imputer has imputed all missing values
print(no_outliers_df.isnull().values.any())
```

```
False
```

In [25]:

```
#create a copy and rename the imputed dataset for clarification
imputed_data_df = no_outliers_df.copy()
```

Potential Limitations of using Imputation to impute missing values:

Imputing values can potentially result in a loss of data; recognising missing data may be valuable - in some cases this information can be utilised by an algorithm. Recognising missing data may be important when data is not missing at random

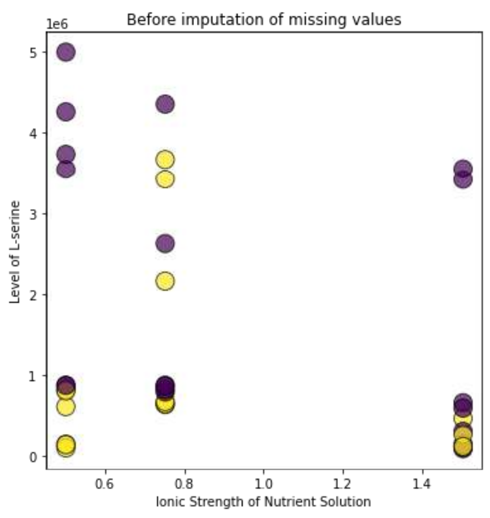
Source: https://towardsdatascience.com/the-ultimate-guide-to-data-cleaning-3969843991d4 (https://towardsdatascience.com/the-ultimate-guide-to-data-cleaning-3969843991d4)

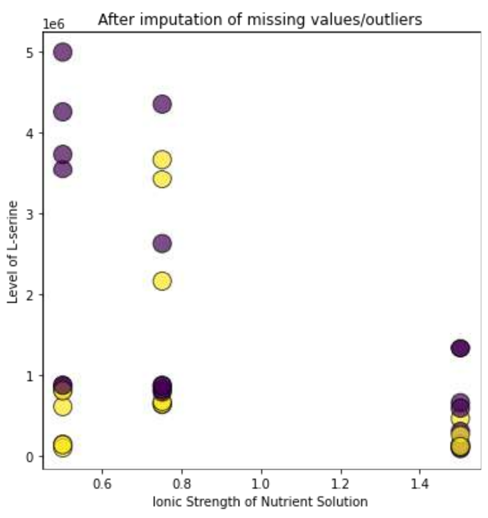
Assess Imputed Data:

In [26]:

```
# #method to color coordinate genotype informed by:
# #https://stackoverflow.com/questions/59232073/scatter-plot-with-3-variables-in-matplotlib
import matplotlib.pyplot as plt
import numpy as np


fig = plt.figure()
fig = plt.figure(figsize=(15, 6))


ax1 = fig.add_subplot(1,2,2)

x = imputed_data_df['Ionic strenght']
y = imputed_data_df['L-serine']
z = imputed_data_df['Genotype']

ax1.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

ax1.set_xlabel('Ionic Strength of Nutrient Solution')
ax1.set_ylabel(f"Level of L-serine")
ax1.set_title("After imputation of missing values/outliers")

ax2 = fig.add_subplot(1,2,1, sharex=ax1, sharey=ax1)

x = sorted_data['Ionic strenght']
y = sorted_data['L-serine']
z = sorted_data['Genotype']

ax2.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

ax2.set_xlabel('Ionic Strength of Nutrient Solution')
ax2.set_ylabel(f"Level of L-serine")
ax2.set_title("Before imputation of missing values")

plt.subplots_adjust(left=0.1,
            bottom=0.1,
            right=0.9,
            top=0.9,
            wspace=0.5,
            hspace=0.4)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```





IterativeImputer seems to have imputed a reasonable new value for ionic strength FS(1.5) (see figure right) - a potential downside of resolving outliers in this way is that it may be removing biological data that is not an outlier but cause by a factor not included in this database - which might offer guidance as to whether ionic strength is influencing compound levels, or another factor

In [27]:

```
cleaned_data = imputed_data_df.copy()
```

Investigating potential bias in the dataset using the FairLens Library

https://fairlens.readthedocs.io/en/stable/user_guide/scorer.html
(https://fairlens.readthedocs.io/en/stable/user_guide/scorer.html)

Assess if imputation has introduced bias into the data:

In [28]:

```
import fairlens as fl
```

In [29]:

```
#sample some data and produce a demographic report
bias_test = ionic_strength_df.iloc[:, :2]
bias_test.index -=1
bias_test['(-)-(4S)-α-terpineol'] = cleaned_data['(-)-(4S)-α-terpineol']
```

```
C:\Users\amyti\AppData\Local\Temp/ipykernel_57556/3980893730.py:4: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  bias_test['(-)-(4S)-α-terpineol'] = cleaned_data['(-)-(4S)-α-terpineol']
```

In [30]:

```
fscorer = fl.FairnessScorer(
    bias_test,
    '(-)-(4S)-α-terpineol',
    ["Genotype", "Ionic strenght"]
)
fscorer.distribution_score()
```

Out[30]:

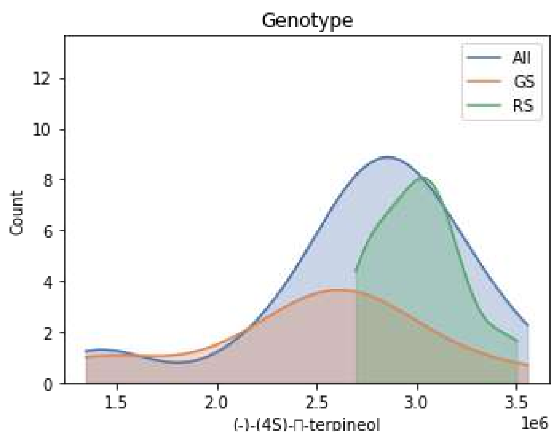
	Group	Distance	Proportion	Counts
0	GS	0.361111	0.500000	18
1	RS	0.361111	0.500000	18
2	FS	0.111111	0.333333	12
3	HS	0.277778	0.333333	12
4	QS	0.333333	0.333333	12
5	GS, FS	0.583333	0.166667	6
6	GS, HS	0.694444	0.166667	6
7	GS, QS	0.222222	0.166667	6
8	RS, FS	0.472222	0.166667	6
9	RS, HS	0.305556	0.166667	6
10	RS, QS	0.500000	0.166667	6

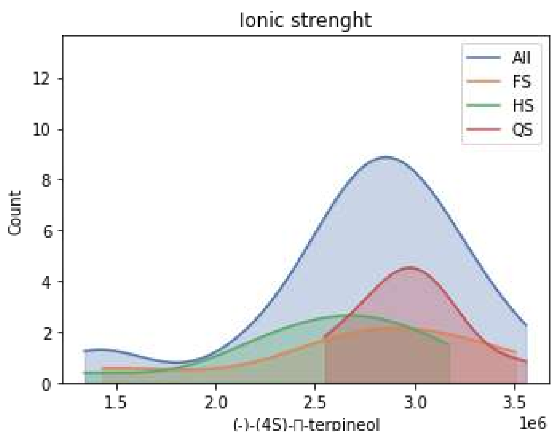
Looking at the demographic report produced by the FairLens library method .distribution_score, the distribution of data seems to be even / unbiased - each genotype has 18 values recorded, each ionic strength interval has 12 values recorded ; combinations of each genotype and ionic strength interval are also evenly distributed (6).

In [31]:

```
fscorer.plot_distributions()
```

```
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_ag
g.py:240: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_ag
g.py:203: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0, flags=flags)
```





The distribution plot helps to clarify the interaction between genotype and compound levels ; looking at the distribution plot on the left, genotype RS seem to have naturally higher '(-)-(4S)-α-terpineol' levels.

In [ ]:

```

```

Strengths and weaknesses of this dataset identified as part of the data cleansing process

Strengths:

- The dataset contains a large number of compounds (over 2000)
- The dataset also specifies the genotype of each entry - this may provide further insight
- The accompanying PDF describes ‘Quality controls (QCs) obtained from pooled samples were used to achieve a higher degree of confidence in annotation using the MS-DIAL 3.98

Weakness:

- Some (approximately 1.5%) of variables have more than 95% of data missing
- Potential outliers in data - outliers may be biologically significant - perhaps an annotation for potential outliers would have offered guidance on whether to remove the potential outlier
- For each compound there are 36 data entries - this is enough to implement a regression model; however a larger number of data entries would allow for a more accure model
- only 3 intervals of ionic strength are tested - ionic strengths greater than 1.5 (full strength) may further increase crop yield

Analysis

First I will identify if that data is normally distributed - this will dictate which statistical tests are appropriate (parametric or non-parametric):

In [32]:

```
import scipy.stats as stats

normal_dist_count = 0
not_normal_dist_count = 0
alpha = 0.5

for column in cleaned_data.iloc[:, 3:]:
    col_data = list(cleaned_data[column].values)
    #check if the column data for each compound is normally distributed
    #if the p value returned by stats.normaltest is lower than 0.5
    if stats.normaltest(col_data)[1] <= alpha:
        normal_dist_count += 1
    else:
        not_normal_dist_count +=1

print(f"Number of columns normally distributed: {normal_dist_count}")
print(f"Number of columns not normally distributed: {not_normal_dist_count}")
```

```
Number of columns normally distributed: 1771
Number of columns not normally distributed: 351
```

As there are a number of columns that are not normally distributed, I will apply non-parametric statistical tests to analyse the dataset

Analysis Technique 1: Kruskal-Wallis Test

Objective: Determine if changes in Ionic Strength result in statistically different metabolic compound levels

My inital research revealed ANOVA test as a potential measure of correlation - however one of the assumptions of ANOVA is that the dependent variable is normally distributed - which is not the case in this data set - further research identified the Kruskal-Wallis test as the non-parametric equivalent to the ANOVA test Source: https://www.statology.org/kruskal-wallis-test-python/ (https://www.statology.org/kruskal-wallis-test-python/)

Kruskal Wallis Assumptions:

1. dependent variable should be measured at the ordinal or continuous level
2. Your independent variable should consist of two or more categorical, independent groups.
3. independence of observations; no relationship between the observations in each group or between the groups themselves.

Source: https://statistics.laerd.com/spss-tutorials/kruskal-wallis-h-test-using-spss-statistics.php (https://statistics.laerd.com/spss-tutorials/kruskal-wallis-h-test-using-spss-statistics.php) Source: https://www.spss-tutorials.com/kruskal-wallis-test/ (https://www.spss-tutorials.com/kruskal-wallis-test/)

I initially implemented the Kruskal Wallis test on all compounds in the table - however this seems too computationally expensive - taking 30+ seconds - I refined my analysis to compounds in the category of the 5 key metabolic markers for plants: the 5 key plant metabolic markers are: (tyrosine, threonine, valine, serine, glutamine, glucose and fructose)
Source:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5025497/#:~:text=Five%20amino%20acids%20(tyrosine%2C%20threonine,%2C%20respectively%20(Steinfath%20et%20al.

In [33]:

```
#retrieve the column names that contain the 5 key metabolic markers
metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

metabol_marker_col_names = []
#retrieve data set column names that match items in metabol_markers
for col in cleaned_data.columns.values:
    for marker in metabol_markers:
        if marker in col:
            metabol_marker_col_names.append(col)
```

Write a function that will perform the Kruskal Wallis Test for each of the compounds listed above:

In [34]:

```
from scipy import stats
#inputs:
#compounds_list: array of compounds to perfrom kruskal wallis test on
#ion_stren_keys: array of ionic strength values (from coolumn in dataset)
#_alpha: significance value (p value level)

#output: array of compound names that have kruskal wallis p value below specified alpha val

def kruskalWallis(compounds_list, ion_stren_keys, _alpha, dataframe):
    try:

        affected_compounds =[]

        for compound in compounds_list:


            data_FS = []
            data_HS = []
            data_QS = []

            compound_data = np.array(dataframe[compound])
            num_key_types = list(set(ion_stren_keys))
            for i in range(ion_stren_keys.size):

                if(ion_stren_keys[i] == num_key_types[0]):
                    data_FS.append(compound_data[i])
                elif (ion_stren_keys[i] == num_key_types[1]):
                    data_HS.append(compound_data[i])
                else:
                    data_QS.append(compound_data[i])

            #perform Kruskal-Wallis Test
            if data_QS:
                if (stats.kruskal(data_FS, data_HS, data_QS).pvalue < _alpha):
                        affected_compounds.append(compound)
            else:
                if (stats.kruskal(data_FS, data_HS).pvalue < _alpha):
                        affected_compounds.append(compound)

        print('Metabolic markers correlated with ionic strength:')
        return affected_compounds

    except KeyError:
      print("Please confirm you are passing a compound name contained in the database")
    except Exception as e:
        print(e)
```

In [35]:

```
ion_stren_keys = np.array(cleaned_data['Ionic strenght'])
#perform kruskal wallis test for list of compounds above 'metabol_marker_col_names'
kruskal_wallis_results_ion = kruskalWallis(metabol_marker_col_names, ion_stren_keys, 0.05,
kruskal_wallis_results_ion
```

```
Metabolic markers correlated with ionic strength:
```

Out[35]:

```
['(-)-jasmonoyl-L-valine',
 '(indol-3-yl)acetyl-L-glutamine',
 'aldehydo-D-glucose',
 'keto-D-fructose',
 'N,N-dihydroxy-L-valine',
 'N-hydroxy-L-valine',
 '1-O-(4-coumaroyl)-β-D-glucose',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 '1-16:0-2-18:3-diacylglycerol-trimethylhomoserine',
 'D-glutamine',
 'L-glucose',
 'L-glutamine',
 'L-serine',
 'oxindole-3-acetyl-L-valine']
```

Interpretation of Kruskal Wallis Test Results: The null hypothesis (H0): The median is equal across all groups. The alternative hypothesis: (Ha): The median is not equal across all groups. If the p-value is less than 0.05, the null hypothesis is rejected. Suggesting that ionic strength affects the compound levels

The results of the Kruskal-Wallis Test indicate that changes in ionic strength result in changes in statistically different metabolic compound levels (listed above)

Limitations of the Kruskal Wallis Method: 'It's important to note that Kruskal-Wallis can only tell us that at least one of the groups originates from a different distribution. It cannot tell us which of the group(s) that is(are).' Source: https://data.library.virginia.edu/getting-started-with-the-kruskal-wallis-test/ (https://data.library.virginia.edu/getting-started-with-the-kruskal-wallis-test/)

A post-hoc test is required to determine which of the groups is statistically different - eg Dunn

Assessment of the accuracy of Kruskal-Wallis results through scatter plots of sample data:

In [36]:

```
kruskal_results = kruskalWallis(metabol_marker_col_names, ion_stren_keys, 0.05, cleaned_dat

#group results by compound type
plot_data = {
    "tyrosine": [],
    "threonine": [],
    "valine": [],
    "serine": [],
    "glutamine": [],
    "glucose": [],
    "fructose":[]
}


metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

for compound in kruskal_results:
    for marker in metabol_markers:
        if (marker in compound):
            plot_data[marker].append(compound)

plot_data
```

```
Metabolic markers correlated with ionic strength:
```

Out[36]:

```
{'tyrosine': [],
 'threonine': [],
 'valine': ['(-)-jasmonoyl-L-valine',
  'N,N-dihydroxy-L-valine',
  'N-hydroxy-L-valine',
  'oxindole-3-acetyl-L-valine'],
 'serine': ['1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
  '1-16:0-2-18:3-diacylglycerol-trimethylhomoserine',
  'L-serine'],
 'glutamine': ['(indol-3-yl)acetyl-L-glutamine', 'D-glutamine', 'L-glutamin
e'],
 'glucose': ['aldehydo-D-glucose',
  '1-O-(4-coumaroyl)-β-D-glucose',
  'L-glucose'],
 'fructose': ['keto-D-fructose']}
```

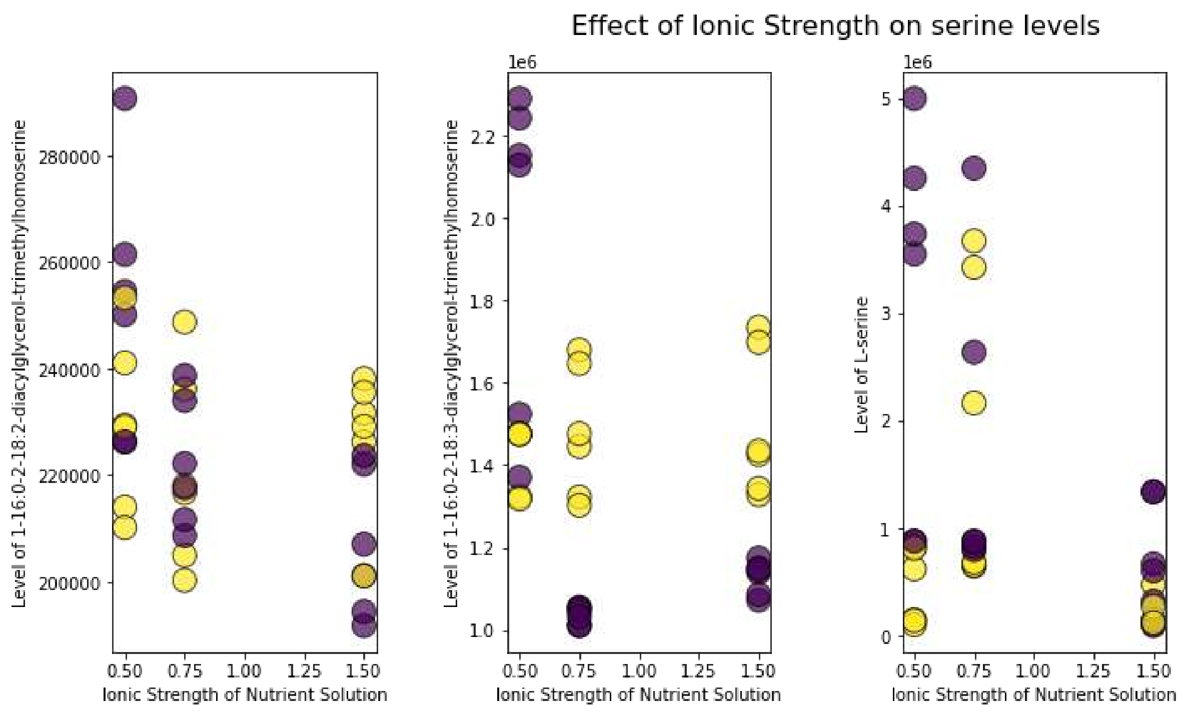
In [37]:

```python
import matplotlib.pyplot as plt

def multiplePlots(values, metabol_marker_name):
    fig = plt.figure()
    fig = plt.figure(figsize=(15, 6))

    i=1
    for compound in values:


        ax1 = fig.add_subplot(1,4,i)

        x = cleaned_data['Ionic strenght']
        y = cleaned_data[compound]
        z = cleaned_data['Genotype']

        ax1.scatter(x, y,
                    linewidths=1, alpha=.7,
                    edgecolor='k',
                    s = 200,
                    c=z)

        ax1.set_xlabel('Ionic Strength of Nutrient Solution')
        ax1.set_ylabel(f"Level of {compound}")

        if i == 5:
            i=1
        else:
            i+=1

    plt.suptitle(f"Effect of Ionic Strength on {metabol_marker_name} levels", fontsize=16)
    plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.5,
                    hspace=0.4)

    plt.show()
```

In [38]:

```
multiplePlots(plot_data['serine'], 'serine')
```

```
<Figure size 432x288 with 0 Axes>
```



The scatter plots of sample data seem to confirm the Kruskal-Wallis results that a change in ionic strength results in a statistical difference in compound levels

Analysis Technique 2: Kendall's Tau Correlation Matrix

Objective: Determine the strength and direction of correlation between each compound and ionic strength

I have chosen to use Kendall's Tau as it is a non-parametric statistical test; my research identified Spearman's Rank as a possible alternative test - however furhter research identified the following:

'Spearman’s is incredibly similar to Kendall’s. It is a non-parametric test that measures a monotonic relationship using ranked data. While it can often be used interchangeably with Kendall’s, Kendall’s is more robust and generally the preferred method of the two. ' Source: https://www.tessellationtech.io/data-science-stats-review/#:~:text=Spearman's%20is%20incredibly%20similar%20to,preferred%20method%20of%20the%20two (https://www.tessellationtech.io/data-science-stats-review/#:~:text=Spearman's%20is%20incredibly%20similar%20to,preferred%20method%20of%20the%20two).

'In the normal case, the Kendall correlation is preferred than the Spearman correlation because of a smaller gross error sensitivity (GES) (more robust) and a smaller asymptotic variance (AV) (more efficient). ' Source: https://www.researchgate.net/post/Does-Spearmans-rho-have-any-advantage-over-Kendalls-tau (https://www.researchgate.net/post/Does-Spearmans-rho-have-any-advantage-over-Kendalls-tau)

Based on these sources I have chosen to implement Kendall's Tau

In [39]:

```
#assemble the dataframe input for Kendall's Tau
corr_prep_df = pd.concat([cleaned_data.iloc[:, :3], cleaned_data[metabol_marker_col_names]]
```

In [40]:

```
correlation_df = corr_prep_df.corr(method='kendall').iloc[:, :2]
```

In [41]:

```
#gather a correlation matrix for the key metabolic markers (included in this dataset)
metabol_corr_df = correlation_df.loc[metabol_marker_col_names]
metabol_corr_df.head()
```

Out[41]:

	Genotype	Ionic strenght
(-)-jasmonoyl-L-valine	-0.243666	0.164980
(indol-3-yl)acetyl-L-glutamine	-0.372144	-0.103592
α-D-glucose 6-phosphate	-0.252326	-0.187851
aldehydo-D-glucose	0.235180	0.438086
keto-D-fructose	0.235180	0.438086

Visualise Kendall's Tau Results using Seaborn Heatmaps:

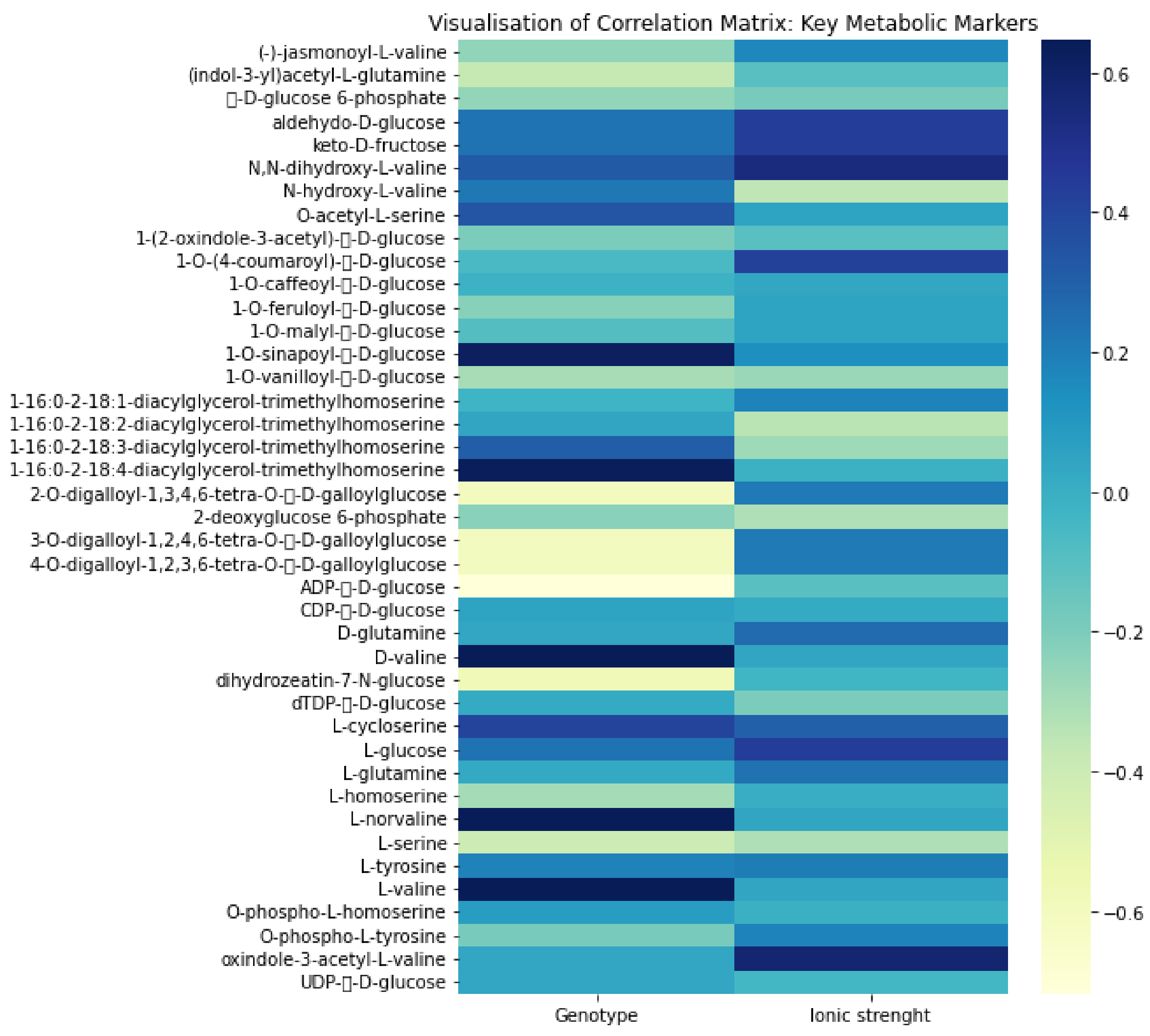
In [42]:

```
import seaborn as sns
f, ax = plt.subplots(figsize=(7, 10))
ax = sns.heatmap(metabol_corr_df, cmap="YlGnBu")
ax.set_title('Visualisation of Correlation Matrix: Key Metabolic Markers')
```

```
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0, flags=flags)
```

Out[42]:

```
Text(0.5, 1.0, 'Visualisation of Correlation Matrix: Key Metabolic Markers')
```



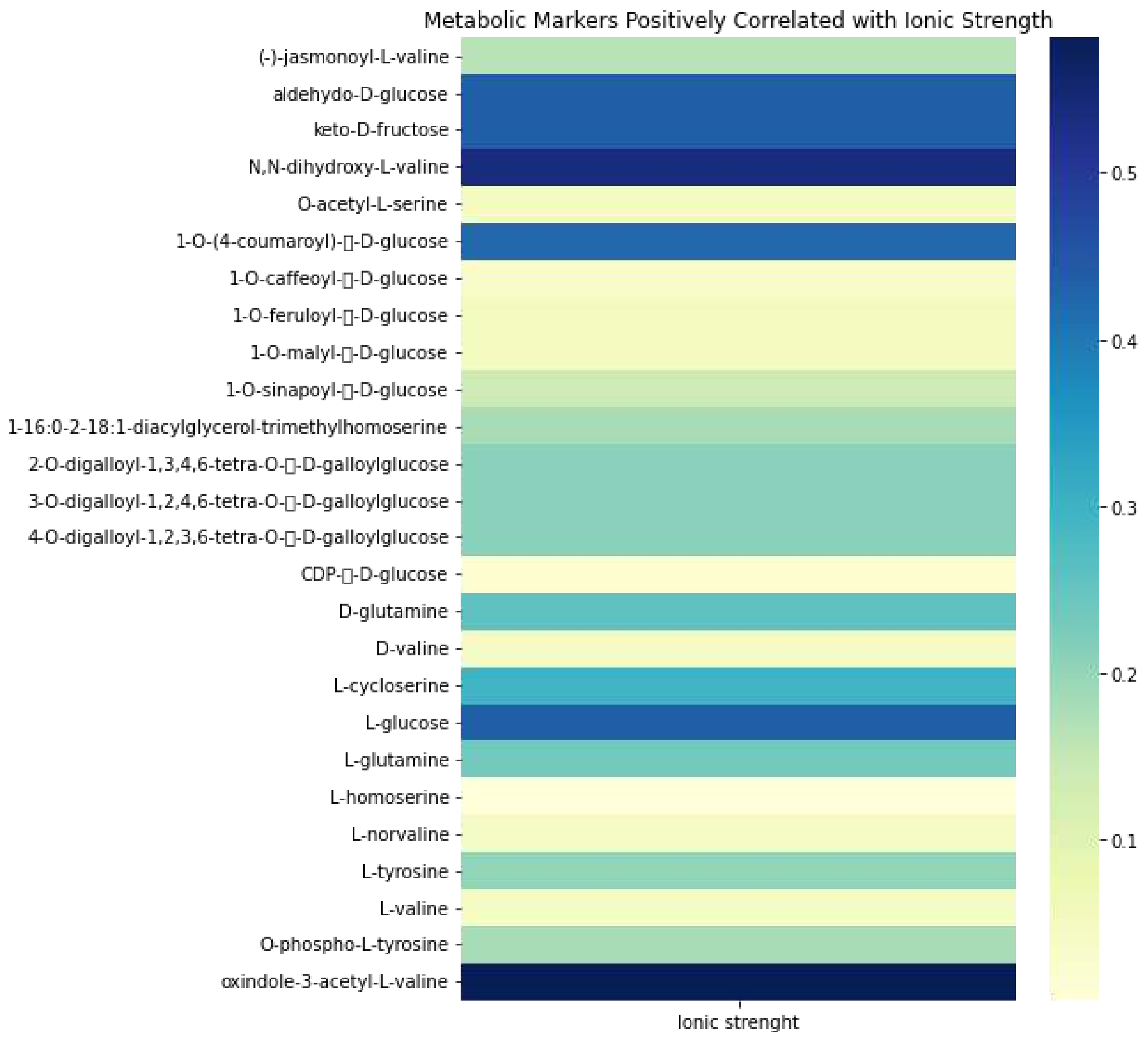
In [43]:

```
#to refine analysis, generate 1 heatmap for compounds positively correlated with ionic stre
#and 1 heatmap for compounds negatively correlated with ionic strength
pos_metabol_corr_df = metabol_corr_df[(metabol_corr_df['Ionic strenght'] > 0)]
pos_metabol_corr_df.drop('Genotype',1, inplace=True)
f, ax = plt.subplots(figsize=(7, 10))
ax = sns.heatmap(pos_metabol_corr_df, cmap="YlGnBu")
ax.set_title('Metabolic Markers Positively Correlated with Ionic Strength')
```

```
C:\Users\amyti\AppData\Local\Temp/ipykernel_57556/2075677102.py:4: FutureWar
ning: In a future version of pandas all arguments of DataFrame.drop except f
or the argument 'labels' will be keyword-only
  pos_metabol_corr_df.drop('Genotype',1, inplace=True)
C:\Users\amyti\anaconda3\lib\site-packages\pandas\core\frame.py:4906: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  return super().drop(
```

Out[43]:

```
Text(0.5, 1.0, 'Metabolic Markers Positively Correlated with Ionic Strengt
h')
```



In [44]:

```
pos_metabol_corr_df.sort_values('Ionic strenght', ascending=False, inplace=True)
#gather the top 5 compounds positively correlated with ionic strength
compound_list = list(pos_metabol_corr_df.index)
top_5_pos_compounds = compound_list[:6]
top_5_pos_compounds
```

```
C:\Users\amyti\anaconda3\lib\site-packages\pandas\util\_decorators.py:311: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  return func(*args, **kwargs)
```

Out[44]:

```
['oxindole-3-acetyl-L-valine',
 'N,N-dihydroxy-L-valine',
 'keto-D-fructose',
 'L-glucose',
 'aldehydo-D-glucose',
 '1-O-(4-coumaroyl)-β-D-glucose']
```

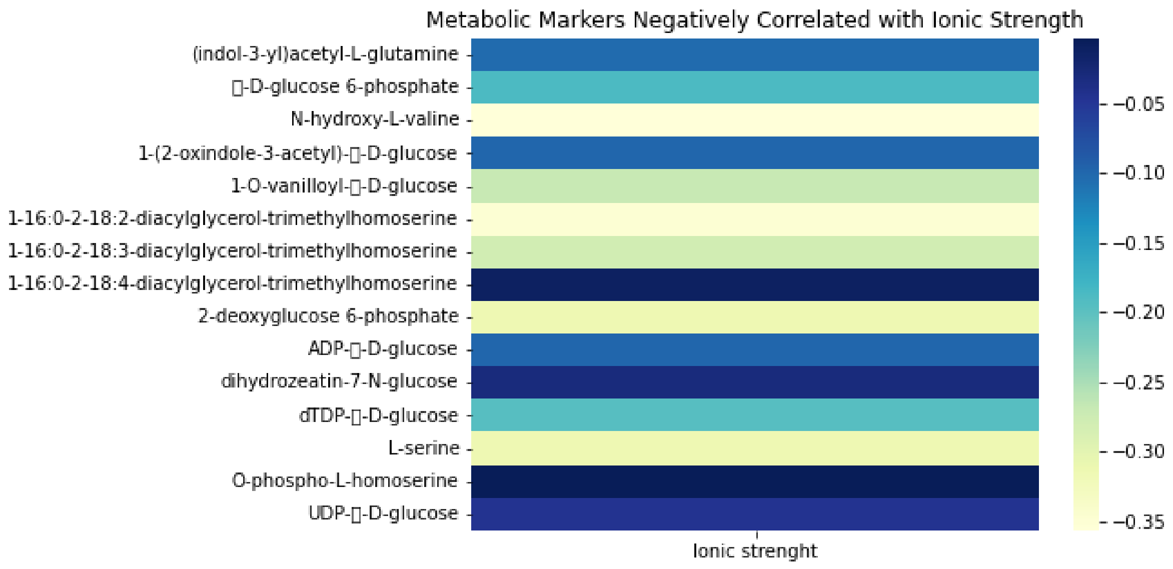
In [45]:

```
#create heatmap without ionic strength
neg_metabol_corr_df = metabol_corr_df[(metabol_corr_df['Ionic strenght'] < 0)]
neg_metabol_corr_df.drop('Genotype',1, inplace=True)
f, ax = plt.subplots(figsize=(7, 5))
ax = sns.heatmap(neg_metabol_corr_df, cmap="YlGnBu")
ax.set_title('Metabolic Markers Negatively Correlated with Ionic Strength')
```

```
C:\Users\amyti\AppData\Local\Temp/ipykernel_57556/277794577.py:3: FutureWarn
ing: In a future version of pandas all arguments of DataFrame.drop except fo
r the argument 'labels' will be keyword-only
  neg_metabol_corr_df.drop('Genotype',1, inplace=True)
```

Out[45]:

```
Text(0.5, 1.0, 'Metabolic Markers Negatively Correlated with Ionic Strengt
h')
```



In [46]:

```
neg_metabol_corr_df.sort_values('Ionic strenght', inplace=True)
#gather the top 5 compounds positively correlated with ionic strength
compound_list = list(neg_metabol_corr_df.index)
top_5_neg_compounds = compound_list[:6]
top_5_neg_compounds
```

Out[46]:

```
['N-hydroxy-L-valine',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 '2-deoxyglucose 6-phosphate',
 'L-serine',
 '1-16:0-2-18:3-diacylglycerol-trimethylhomoserine',
 '1-O-vanilloyl-β-D-glucose']
```

According to
http://polisci.usca.edu/apls301/Text/Chapter%2012.%20Significance%20and%20Measures%20of%20Associatior
(http://polisci.usca.edu/apls301/Text/Chapter%2012.%20Significance%20and%20Measures%20of%20Associatio
Kendall Tau B values can be interpreted as follows:

Less than + or - 0.10: very weak

+ or -0.10 to 0.19: weak
+ or - 0.20 to 0.29: moderate
+ or - 0.30 or above: strong

In [47]:

```
#identify which correlations are below -0.30
neg_results = metabol_corr_df.loc[metabol_corr_df['Ionic strenght'] <= -0.30]
neg_results = neg_results.style.set_caption("Metabolic Markers with a strong negative corre
neg_results
```

Out[47]:

Metabolic Markers with a strong negative correlation with Ionic Strength

	Genotype	Ionic strenght
N-hydroxy-L-valine	0.216912	-0.356534
1-16:0-2-18:2-diacylglycerol-trimethylhomoserine	0.039873	-0.349144
2-deoxyglucose 6-phosphate	-0.230375	-0.314613
L-serine	-0.398726	-0.314613

In [48]:

```
#identify which correlations are above 0.30
pos_results = metabol_corr_df.loc[metabol_corr_df['Ionic strenght'] >= 0.30]
pos_results = pos_results.style.set_caption("Metabolic Markers with a strong positive corre
pos_results
```

Out[48]:

Metabolic Markers with a strong positive correlation with Ionic Strength

	Genotype	Ionic strenght
aldehydo-D-glucose	0.235180	0.438086
keto-D-fructose	0.235180	0.438086
N,N-dihydroxy-L-valine	0.318981	0.537144
1-O-(4-coumaroyl)-β-D-glucose	-0.057594	0.422042
L-glucose	0.235180	0.438086
oxindole-3-acetyl-L-valine	0.035527	0.580735

The results of Kendall's Tau indicate that the metabolic markers listed above (see tables) have a strong correlation with ionic strength

L-serine and N,N-dihydroxy-L-valine seem to be the only metabolic markers that have a strong correlation with genotype.

In [49]:

```
#create a list kendall tau results (list of compounds that have a strong correlation with i
#in order to cross validate results with kruskal wallis results
kendall_results = list(neg_results.index) + list(pos_results.index)
kendall_results
```

Out[49]:

```
['N-hydroxy-L-valine',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 '2-deoxyglucose 6-phosphate',
 'L-serine',
 'aldehydo-D-glucose',
 'keto-D-fructose',
 'N,N-dihydroxy-L-valine',
 '1-O-(4-coumaroyl)-β-D-glucose',
 'L-glucose',
 'oxindole-3-acetyl-L-valine']
```

Cross validation of Kendall's Tau and Kruskal Wallis results

In [50]:

```
validate_results = [x for x in kruskal_wallis_results_ion if x in kendall_results]
validate_results
```

Out[50]:

```
['aldehydo-D-glucose',
 'keto-D-fructose',
 'N,N-dihydroxy-L-valine',
 'N-hydroxy-L-valine',
 '1-O-(4-coumaroyl)-β-D-glucose',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 'L-glucose',
 'L-serine',
 'oxindole-3-acetyl-L-valine']
```

The results of the 2 statistical tests show that the metabolic markers listed above are strongly correlated with ionic strength

Evaluation of techniques:

In [51]:

```
len(kruskal_wallis_results_ion)-len(kendall_results)
```

Out[51]:

```
4
```

In [52]:

```
len(validate_results)
```

Out[52]:

```
9
```

The kruskal wallis test results returned 14 metabolic compounds; the kendall's tau test returned 10 metabolic compounds; 9 of the 14 metabolic compounds were cross validated. This identifies some discrepancies between the 2 statistical tests - however the majority of metabolic compounds were cross validated.

Analysis Technique 3: Random Forest Regressor to assess feature importance

In [53]:

```
validate_results
```

Out[53]:

```
['aldehydo-D-glucose',
 'keto-D-fructose',
 'N,N-dihydroxy-L-valine',
 'N-hydroxy-L-valine',
 '1-O-(4-coumaroyl)-β-D-glucose',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 'L-glucose',
 'L-serine',
 'oxindole-3-acetyl-L-valine']
```

In [54]:

```
#gather the original categorical variables:
categorical_df = non_bias_df[['Genotype','Ionic strenght']].copy()
categorical_df.head()
```

Out[54]:

	Genotype	Ionic strenght
1	GS	FS
2	GS	FS
3	GS	FS
4	GS	FS
5	GS	FS

In [55]:

```
random_forest_prep = pd.get_dummies(categorical_df)
random_forest_prep.index -=1
```

In [56]:

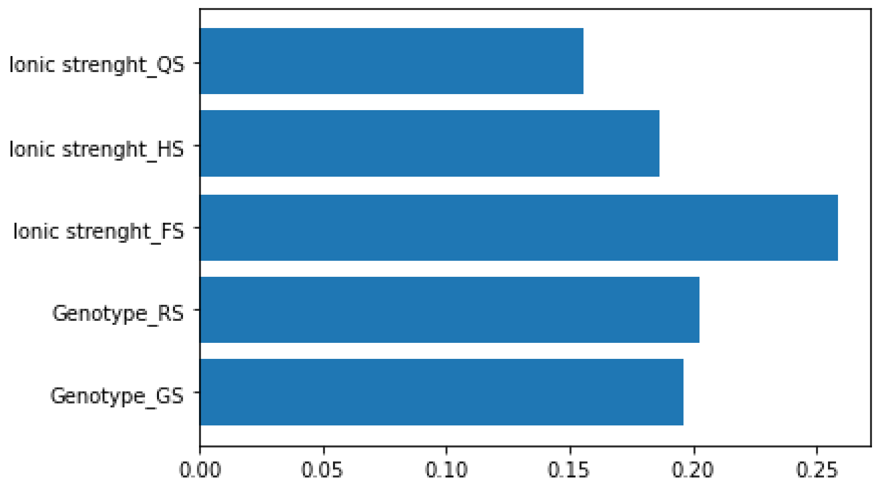
```
from sklearn.ensemble import RandomForestRegressor

def featureImportance(colName, dummy_df):
    x = dummy_df
    y= cleaned_data[colName]
    #method below found at: https://mljar.com/blog/feature-importance-in-random-forest/
    rf = RandomForestRegressor()
    rf.fit(x, y)
    print(rf.feature_importances_)
    plt.barh(random_forest_prep.columns, rf.feature_importances_)
```

Visual Evaluation of Random Forest Regressor Technique:

In [57]:

```
featureImportance('1-O-(4-coumaroyl)-β-D-glucose', random_forest_prep)
```

```
[0.19614521 0.20297879 0.25865229 0.1868832  0.15534052]
```

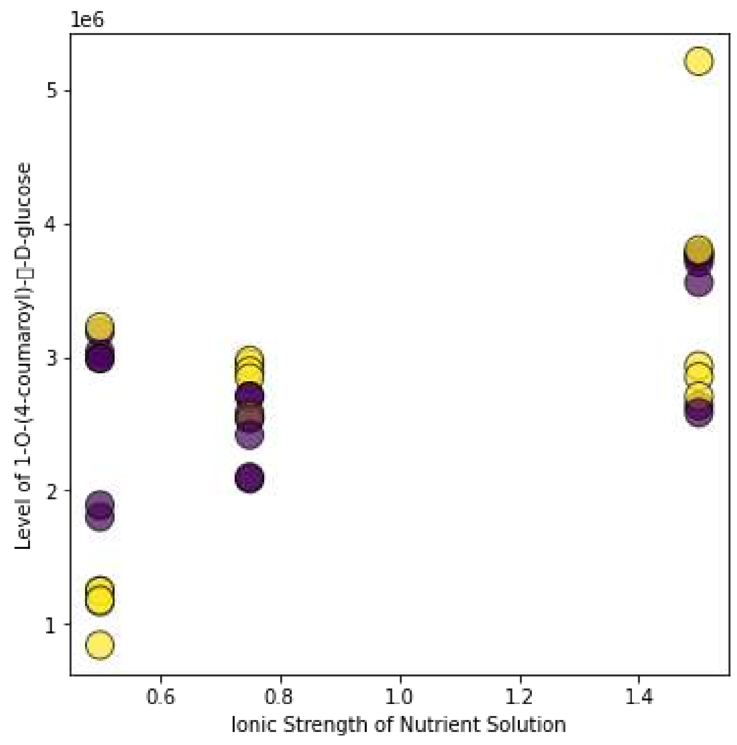


In [58]:

```
x = cleaned_data['Ionic strenght']
y = cleaned_data['1-O-(4-coumaroyl)-β-D-glucose']
z = cleaned_data['Genotype']

fig = plt.figure(figsize=(6, 6))
plt.scatter(x, y,
           linewidths=1, alpha=.7,
           edgecolor='k',
           s = 200,
           c=z)

plt.title("1-O-(4-coumaroyl)-β-D-glucose levels relative to Ionic Strength of Nutrient Solu
plt.xlabel("Ionic Strength of Nutrient Solution")
plt.ylabel("Level of 1-O-(4-coumaroyl)-β-D-glucose")
plt.show()
```

```
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0, flags=flags)
```

1-O-(4-coumaroyl)-▯-D-glucose levels relative to Ionic Strength of Nutrient Solution and Genotype



Analysing the 2 figures above, the random forest regressor technnique identified that full ionic strength of nutrient solution had the most impact on 1-O-(4-coumaroyl)-β-D-glucose - the scatter plot shows that full ionic strength resulted in higher compound levels. The random forest regressor indicates that the GS genotype is the next most influential factor; I feel that there is some ambiguity as to wheter this is reflected in the corresponding scatter plot - as an alternative approach I will identified which interval of ionic strength resulted in the highest mean peak value.

Exploration of optimal ionic strength level

To assess which level of ionic strength results in higher metabolic marker levels overall, I will calculate the mean compound level at each ionic strength interval. I will then determine which category produces the highest levels for each compound:

In [59]:

```
#define a function that can be utilised by all 3 datasets
from statistics import mean
def meanByCategory(dataframe,compounds_list, category_keys):

    mean_by_compounds = []
    for compound in compounds_list:
            data_FS = []
            data_HS = []
            data_QS = []

            compound_data = np.array(dataframe[compound])
            num_key_types = list(set(category_keys))
            for i in range(category_keys.size):

                if(category_keys[i] == num_key_types[0]):
                    data_FS.append(compound_data[i])
                elif (category_keys[i] == num_key_types[1]):
                    data_HS.append(compound_data[i])
                else:
                    data_QS.append(compound_data[i])


            if data_QS:
                mean_by_compounds.append({
                    "Compound": compound,
                    f"{num_key_types[0]}": mean(data_FS),
                    f"{num_key_types[1]}": mean(data_HS),
                    f"{num_key_types[2]}": mean(data_QS),
                })
            else:
                mean_by_compounds.append({
                    "Compound": compound,
                    f"{num_key_types[0]}": mean(data_FS),
                    f"{num_key_types[1]}": mean(data_HS),
                })

    return mean_by_compounds
```

In [60]:

```
ion_stren_keys = np.array(cleaned_data['Ionic strenght'])
mean_by_compounds = meanByCategory(cleaned_data ,metabol_marker_col_names, ion_stren_keys)
#average compound level for each ionic strength interval
mean_by_compounds
```

Out[60]:

```
[{'Compound': '(-)-jasmonoyl-L-valine',
  '0.5': 1614247.8333333333,
  '0.75': 2255507.2323235967,
  '1.5': 2043532.2979794967},
 {'Compound': '(indol-3-yl)acetyl-L-glutamine',
  '0.5': 1777116.0833333333,
  '0.75': 554687.8088243696,
  '1.5': 1180055.6666666667},
 {'Compound': 'α-D-glucose 6-phosphate',
  '0.5': 2533627,
  '0.75': 2140893,
  '1.5': 1912881},
 {'Compound': 'aldehydo-D-glucose',
  '0.5': 10461299.766666658,
  '0.75': 15551448.76666652,
  '1.5': 14871553.266666949},
 {'Compound': 'keto-D-fructose',
  '0.5': 10461299.766666658,
```

Find highest mean for each compound:

In [61]:

```
def tallyHighestMeans(mean_data, tally_categories, categoryName):
    #remove 'Compound' key so that the following code can compare numeric means to find hig
    #do not affect in place - compound labels may be needed for analysis
    mean_data_copy = mean_data
    for x in mean_data_copy:
        x.pop('Compound', None)

    FS_count = 0
    HS_count = 0
    QS_count = 0

    for x in mean_data_copy:
        if max(x, key=x.get) == tally_categories[0]:
            FS_count +=1
        elif max(x, key=x.get) == tally_categories[1]:
            HS_count +=1
        elif len(tally_categories) ==3 and max(x, key=x.get) == tally_categories[2]:
            QS_count +=1

    print(f"Number of compounds with highest mean at {tally_categories[0]} {categoryName}:
    print(f"Number of compounds with highest mean at {tally_categories[1]} {categoryName}:
    #if there is a third category
    #(Light intensity dataset has 2 categories)
    if QS_count != 0:
        print(f"Number of compounds with highest mean at {tally_categories[2]} {categoryNam
```

In [62]:

```
tallyHighestMeans(mean_by_compounds, ['0.5','0.75','1.5'], 'Ionic Strength')
```

```
Number of compounds with highest mean at 0.5 Ionic Strength: 11
Number of compounds with highest mean at 0.75 Ionic Strength: 11
Number of compounds with highest mean at 1.5 Ionic Strength: 19
```

The results above indicate that the full ionic strength optimises the most metabolic markers (Full ionic strength defined as 1.5 ± 0.1dS m−1'

Potential Limitations of this method:
The mean can be influenced by outliers and skewed distributions - this has the potential to distort conclusions.
Source: https://dzone.com/articles/limitations-of-the-measure-of-central-tendency-sta#:~:text=Limitations%20of%20the%20Mean%3A,by%20outliers%20and%20skewed%20distributions (https://dzone.com/articles/limitations-of-the-measure-of-central-tendency-sta#:~:text=Limitations%20of%20the%20Mean%3A,by%20outliers%20and%20skewed%20distributions).

Investigation of Light Strength on Metabolic Markers

Exploration of Dataset 2:

In [63]:

```
#dataset was downloaded as excel file (.xlsx)
raw_data_light = pd.read_excel('Dataset Metabolomics.xlsx', sheet_name=2)
```

2. Data Cleansing

In [64]:

```
#convert the initial read of data into a pandas dataframe ; pandas dataframe presents data
#provides utility functions to access/edit the data
raw_data_light_df = pd.DataFrame(raw_data_light)
```

In [65]:

```
light_strength_df = formatData(raw_data_light_df)
light_strength_df
```

Out[65]:

	Genotype	Light Intensity	CPDs ID	(-)-(4S)-α-terpineol	(-)-α-amorphene	(-)-α-bisabolol	(-)-α-cuprenene	(-)-α-pinene	caryophy
1	GS	FL	40-r001	1205077	NaN	431898	NaN	555694	
2	GS	FL	40-r002	1410285	NaN	431933	NaN	660584	
3	GS	FL	41-r001	1662857	NaN	NaN	NaN	771983	
4	GS	FL	41-r002	1610443	NaN	NaN	NaN	741632	
5	GS	FL	42-r001	1037073	NaN	171701	NaN	482233	
6	GS	FL	42-r002	1023349	NaN	NaN	NaN	466195	
7	GS	LL	46-r001	1138373	522153	166031	522153	539752	52
8	GS	LL	46-r002	1149601	550745	174644	550745	543641	55
9	GS	LL	47-r001	2283061	1433428	NaN	1433428	1052616	143
10	GS	LL	47-r002	2308304	1429244	NaN	1429244	1034627	142
11	GS	LL	48-r001	1257236	NaN	315758	NaN	825899	
12	GS	LL	48-r002	1270991	NaN	320125	NaN	654130	
13	RS	FL	37-r001	1777933	NaN	281959	NaN	806839	
14	RS	FL	37-r002	1787412	NaN	279694	NaN	809770	
15	RS	FL	38-r001	1210336	NaN	294633	NaN	631454	
16	RS	FL	38-r002	1197336	NaN	306021	NaN	606495	
17	RS	FL	39-r001	1044233	NaN	187080	NaN	525831	
18	RS	FL	39-r002	1051849	NaN	183880	NaN	532138	
19	RS	LL	43-r001	1661456	NaN	280847	NaN	800649	
20	RS	LL	43-r002	1669311	NaN	287609	NaN	784380	
21	RS	LL	44-r001	1444719	NaN	NaN	NaN	673174	

	Genotype	Light Intensity	CPDs ID	(-)-(4S)-α-terpineol	(-)-α-amorphene	(-)-α-bisabolol	(-)-α-cuprenene	(-)-α-pinene	caryophy
22	RS	LL	44-r002	1437332	NaN	NaN	NaN	683960	
23	RS	LL	45-r001	2055687	NaN	NaN	NaN	967116	
24	RS	LL	45-r002	2055721	NaN	NaN	NaN	970036	

24 rows × 3876 columns

Flaw of Dataset: Ionic Strength dataset contains 3980 columns columns, the light strength dataset contains 3876 columns - this inconsistency suggests that this dataset is missing some compounds. The ionic strength data set contains 36 entries for each compound, the light strength dataset contains 24 entries for each compound ; the number of entries for light strength dataset is 2/3 the size - a smaller number of data entries may reduce accuracy of statistical analysis.

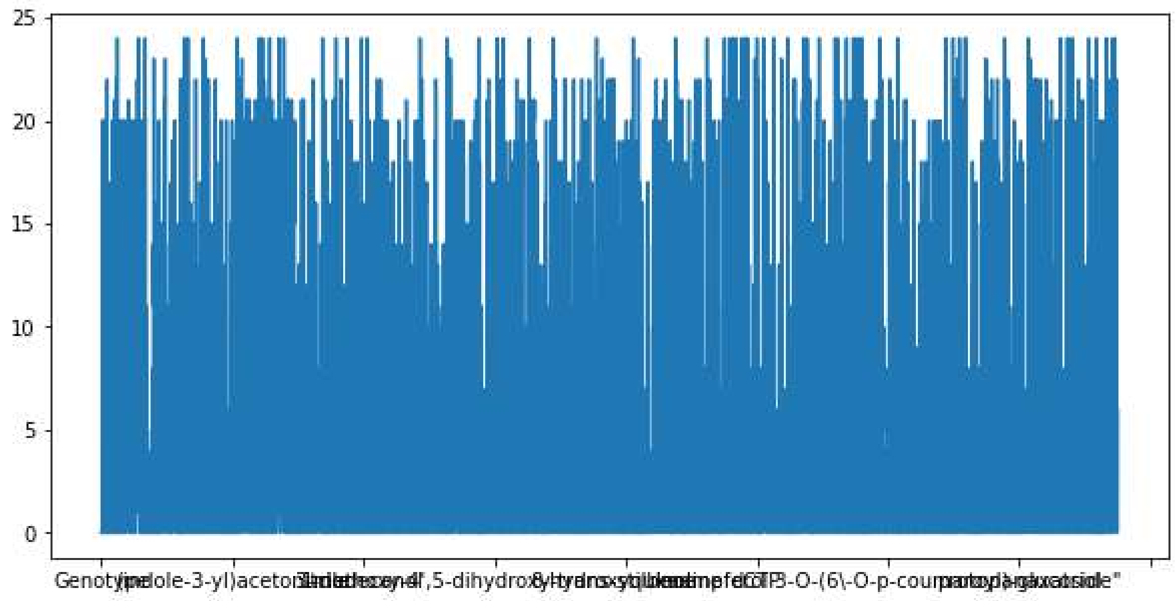
Explore levels of missing Data

In [66]:

```
missing = light_strength_df.isnull().sum()
missing.plot(figsize=(10,5))
```

Out[66]:

```
<AxesSubplot:>
```



Remove varibales with more than 10% of data missing - to reduce bias in the dataset:

In [67]:

```
non_bias_df_light = removeBiasVar(light_strength_df)
```

```
Remaining % of Data: 52.956364575264644
```

Prepare Data for Outlier Detection and removal:

In [68]:

```
#implement dummy variables ready for Iterative imputer
#otherwise the iterative imputer returns the following error:
#ValueError: could not convert string to float: 'GS'

#map ionic strength types to numbers so that rows can be grouped using sort_values (ready f
#outlier detection is being done within each ionic strength band (FS,HS,QS)
non_bias_dummy_df = non_bias_df_light.copy()
non_bias_dummy_df['Genotype'] = non_bias_dummy_df['Genotype'].map({'GS': 1, 'RS':0})
non_bias_dummy_df['Light Intensity'] = non_bias_dummy_df['Light Intensity'].map({'FL': 1, '
```

In [69]:

```
#remove any duplicate columns names
#https://stackoverflow.com/questions/14984119/python-pandas-remove-duplicate-columns
non_bias_dummy_df = non_bias_dummy_df.loc[:,~non_bias_dummy_df.columns.duplicated()].copy()
non_bias_dummy_df.head()
```

Out[69]:

	Genotype	Light Intensity	CPDs ID	(-)-(4S)-α-terpineol	(-)-α-pinene	(-)-β-phellandrene	(-)-β-pinene	(-)-endo-fenchol	(-)-5'demethylyatei
1	1	1	40-r001	1205077	555694	555694	555694	1205077	Nal
2	1	1	40-r002	1410285	660584	660584	660584	1410285	87122!
3	1	1	41-r001	1662857	771983	771983	771983	1662857	11700
4	1	1	41-r002	1610443	741632	741632	741632	1610443	130506
5	1	1	42-r001	1037073	482233	482233	482233	1037073	56157

5 rows × 2050 columns

Type *Markdown* and LaTeX: α^2

Detecting Potential Outliers using the IQR method

In [70]:

```
outliers_df = non_bias_dummy_df.copy()
```

Group Ionic strength rows together: for calculation of outliers within each subgroup

In [71]:

```
sorted_data = outliers_df.sort_values('Light Intensity', inplace=False)
#use the drop paramter to prevent old index system being added as a column
sorted_data.reset_index(drop=True, inplace=True)
```

In [72]:

```
no_outliers_df = removeOutliers(sorted_data.columns[3:], sorted_data, 'Light Intensity', [1
```

In [73]:

```
##confirm that outliers have been removed / the dataset has been modified
sorted_data.equals(no_outliers_df)
```

Out[73]:

```
False
```

In [74]:

```
# explicitly require this experimental feature
from sklearn.experimental import enable_iterative_imputer  # noqa
# now you can import normally from sklearn.impute
from sklearn.impute import IterativeImputer

def imputedValues(list_col_names, dataframe, independentVar):
    for compound in col_names:
        if dataframe[f"{compound}"].isnull().values.any():
            temp_df = dataframe[['Genotype', independentVar]].copy()
            temp_df[f"{compound}"] = dataframe[f"{compound}"]
            #index starts at 1
            imputer = IterativeImputer()
            temp_df = imputer.fit_transform(temp_df)
            #iterative imputer returns a numpy.ndarray
            #convert numpy.ndarray to pandas dataframe
            #for easier updating of data in table (repalce data with results from iterative
            temp_pandas_df = pd.DataFrame(temp_df, columns=['Genotype', independentVar, f"{
            #index starts at 0 - match the index of the original dataframe
            #otherwise last line of data is lost
            #temp_pandas_df.index += 1
            #update the values in the table to include the imputed values
            dataframe[f"{compound}"] = temp_pandas_df[f"{compound}"]

    return dataframe
```

In [75]:

```
#create a list of all chemical compound columns in dataframe
col_names = list(no_outliers_df)
#remove the first three column names (Genotype, ionic strength and CPD ID - as they are not
col_names = col_names[3:]

imputed_df = imputedValues(col_names, no_outliers_df, 'Light Intensity')
```

In [76]:

```
#confirm that Iterative Imputer has imputed all missing values
print(imputed_df.isnull().values.any())
```

```
False
```

2. Analysis

2. Analysis Technique 1: Kruskal-Wallis Test

In [77]:

```
#retrieve the column names that contain the 5 key metabolic markers
metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

metabol_marker_col_names = []

for col in imputed_df.columns.values:
    for marker in metabol_markers:
        if marker in col:
            metabol_marker_col_names.append(col)
```

In [78]:

```
light_stren_keys = np.array(imputed_df['Light Intensity'])
kruskal_wallis_results_light = kruskalWallis(metabol_marker_col_names, light_stren_keys, 0.
print('Metabolic markers correlated with light strength:')
kruskal_wallis_results_light
```

```
Metabolic markers correlated with ionic strength:
Metabolic markers correlated with light strength:
```

Out[78]:

```
['α-D-glucose 1-methylene-phosphonate',
 'N,N-dihydroxy-L-tyrosine',
 '1-O-(4-coumaroyl)-β-D-glucose',
 '1-O,6-O-digalloyl-β-D-glucose',
 '1,2-di-O-sinapoyl-β-D-glucose',
 '2-O-digalloyl-1,3,4,6-tetra-O-β-D-galloylglucose',
 '3-O-digalloyl-1,2,4,6-tetra-O-β-D-galloylglucose',
 '4-(indol-3-yl)butanoyl-β-D-glucose',
 '4-O-digalloyl-1,2,3,6-tetra-O-β-D-galloylglucose',
 'L-cycloserine',
 'L-homoserine',
 'oxindole-3-acetyl-L-valine',
 'UDP-α-D-glucose']
```

Kruskal_Wallis identifies that the changes in light intensity result in statistically different levels of the compounds listed above

2. Analysis Technique 2: Kendall's Tau Correlation Matrix

Kendall-Tau Correlation Matrix:

In [79]:

```
#retrieve the column names that contain the 5 key metabolic markers in plants
metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

col_names = []
for x in imputed_df.columns.values[3:]:
    for marker in metabol_markers:
        if marker in x:
            col_names.append(x)
```

In [80]:

```
corr_prep_df = pd.concat([imputed_df.iloc[:, :3], imputed_df[col_names]], axis="columns")
```

In [81]:

```
correlation_df = corr_prep_df.corr(method='kendall').iloc[:, :2]
```

In [82]:

```
#gather a correlation matrix for the key metabolic markers (included in this dataset)
metabol_corr_df_light = correlation_df.loc[col_names]
metabol_corr_df_light
```

Out[82]:

	Genotype	Light Intensity
(-)-jasmonoyl-L-valine	-0.391254	0.160514
(indol-3-yl)acetyl-L-glutamine	-0.110354	-0.310997
α-D-glucose 1-methylene-phosphonate	-0.140450	-0.722315
α-D-glucose 6-phosphate	-0.260836	-0.140450
α-methyl-tyrosine	-0.412066	-0.180907
N,N-dihydroxy-L-valine	0.211058	-0.241209
N,N-dihydroxy-L-tyrosine	-0.150482	-0.561801
N<sup>5</sup>-methyl-L-glutamine	0.180579	-0.310997
O-acetyl-L-serine	0.321029	0.020064
1-O-(4-coumaroyl)-β-D-glucose	0.211058	0.482418
1-O-feruloyl-β-D-glucose	0.130893	-0.281924
1-O-sinapoyl-β-D-glucose	0.292527	-0.272352
1-O-vanilloyl-β-D-glucose	-0.240772	-0.140450
1-16:0-2-18:3-diacylglycerol-trimethylhomoserine	0.160806	0.291461
1-16:0-2-18:4-diacylglycerol-trimethylhomoserine	0.471511	-0.200643
1-O,6-O-digalloyl-β-D-glucose	0.241209	0.482418
1,2-di-O-sinapoyl-β-D-glucose	0.100322	0.682187
2-O-digalloyl-1,3,4,6-tetra-O-β-D-galloylglucose	-0.682187	-0.401286
2-deoxyglucose 6-phosphate	-0.230740	-0.250804
3-O-digalloyl-1,2,4,6-tetra-O-β-D-galloylglucose	-0.682187	-0.401286
4-(indol-3-yl)butanoyl-β-D-glucose	-0.200643	0.371190
4-O-digalloyl-1,2,3,6-tetra-O-β-D-galloylglucose	-0.682187	-0.401286
4-phosphooxy-L-threonine	0.080403	-0.190957
6-O-galloylglucose	-0.251718	0.322198
CDP-α-D-glucose	-0.100322	0.250804
D-glutamine	0.240772	0.300965
D-valine	0.722315	0.040129
dTDP-α-D-glucose	-0.120386	0.240772
L-cycloserine	0.541736	0.441415
L-glutamine	0.240772	0.300965
L-homoserine	-0.120386	-0.371190
L-norvaline	0.722315	0.040129

	Genotype	Light Intensity
L-tyrosine	0.421350	0.321029
L-valine	0.722315	0.040129
O-phospho-L-homoserine	-0.220707	0.250804
O-phospho-L-tyrosine	-0.190611	-0.160514
oxindole-3-acetyl-L-valine	-0.100322	-0.461479
salicylate β-D-glucose ester	0.200643	0.060193
UDP-α-D-glucose	-0.120386	0.351125

Exploration of optimal light strength level

In [83]:

```
#function meanByCategory requires string keys for each category
def map_light_category(x):
    if x == 0:
        return 'LL'
    else:
        return 'FL'
```

In [84]:

```
mapped_light_stren_keys = map(map_light_category, light_stren_keys)
mapped_light_stren_keys = list(mapped_light_stren_keys)
mapped_light_stren_keys = np.array(mapped_light_stren_keys)
```

In [85]:

```
mean_by_compounds = meanByCategory(imputed_df,metabol_marker_col_names, mapped_light_stren_
```

In [86]:

```
tallyHighestMeans(mean_by_compounds, ['FL','LL'], 'Light Intensity')
```

```
Number of compounds with highest mean at FL Light Intensity: 20
Number of compounds with highest mean at LL Light Intensity: 19
```

The results indicate that full light intensity optimises the most metabolic markers - the difference between number of optimised compounds is less than ionic strength - a change in ionic strength affected a larger number of compounds.

Investigation of Effect of Nutrient Solution Type on Metabolic Markers

Exploration of Dataset 3:

In [87]:

```
#dataset was downloaded from link above
raw_data_macro = pd.read_excel('Dataset Metabolomics.xlsx', sheet_name=1)
```

3. Data Cleansing

In [88]:

```
#convert the initial read of data into a pandas dataframe ; pandas dataframe presents data
#provides utility functions to access/edit the data
raw_data_macro_df = pd.DataFrame(raw_data_macro)
```

In [89]:

```
macro_data_df = formatData(raw_data_macro_df)
macro_data_df.head()
```

Out[89]:

	Genotype	Nutrient solution	CPDs ID	(-)-(4S)-α-terpineol	(-)-α-amorphene	(-)-α-bisabolol	(-)-α-cuprenene	(-)-α-pinene	(-) caryophylle
1	GS	SCa	28-r001	1107066	NaN	638934	NaN	1669992	N
2	GS	SCa	28-r002	1097978	NaN	628834	NaN	544200	N
3	GS	SCa	29-r001	1952450	NaN	689708	NaN	1397469	N
4	GS	SCa	29-r002	1939225	NaN	683046	NaN	1398572	N
5	GS	SCa	30-r001	2055740	NaN	651042	NaN	508343	N

5 rows × 4055 columns

Flaw of Dataset: Ionic Strength dataset contains 3980 columns columns, the light strength dataset contains 3876 columns - this inconsistency suggests that this dataset is missing some compounds. The ionic strength data set contains 36 entries for each compound, the light strength dataset contains 24 entries for each compound ; the number of entries for light strength dataset is 2/3 the size - a smaller number of data entries may reduce accuracy of statistical analysis.

Explore levels of missing Data

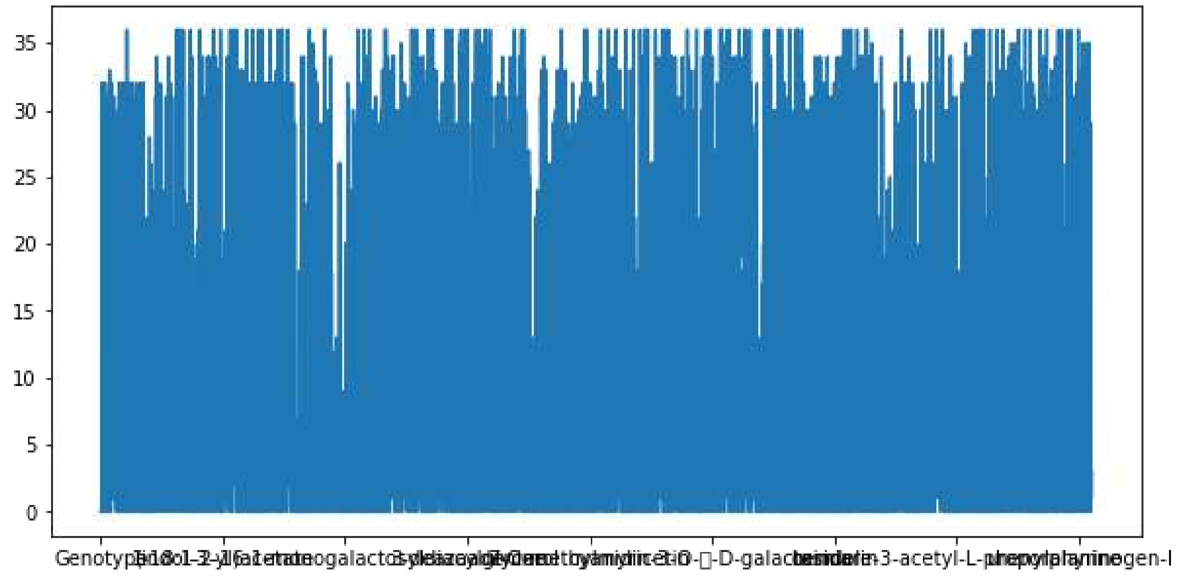
In [90]:

```
missing = macro_data_df.isnull().sum()
missing.plot(figsize=(10,5))
```

Out[90]:

```
<AxesSubplot:>
```

```
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0, flags=flags)
```



Remove varibales with more than 10% of data missing - to reduce bias in the dataset:

In [91]:

```
non_bias_df_macro = removeBiasVar(macro_data_df)
```

```
Remaining % of Data: 47.92694965449161
```

Prepare Data for Outlier Detection and removal:

In [92]:

```
#implement dummy variables ready for Iterative imputer
#otherwise the iterative imputer returns the following error:
#ValueError: could not convert string to float: 'GS'

#map ionic strength types to numbers so that rows can be grouped using sort_values (ready f
#outlier detection is being done within each ionic strength band (FS,HS,QS)
non_bias_dummy_df = non_bias_df_macro.copy()
non_bias_dummy_df['Genotype'] = non_bias_dummy_df['Genotype'].map({'GS': 1, 'RS':0})
non_bias_dummy_df['Nutrient solution'] = non_bias_dummy_df['Nutrient solution'].map({'SCa':
```

In [93]:

```
#remove any duplicate columns names
#https://stackoverflow.com/questions/14984119/python-pandas-remove-duplicate-columns
non_bias_dummy_df = non_bias_dummy_df.loc[:,~non_bias_dummy_df.columns.duplicated()].copy()
non_bias_dummy_df.head()
```

Out[93]:

	Genotype	Nutrient solution	CPDs ID	(-)-(4S)-α-terpineol	(-)-α-bisabolol	(-)-α-pinene	(-)-β-phellandrene	(-)-β-pinene	(-)-endo-fenchol	p
1	1	0	28-r001	1107066	638934	1669992	1669992	1669992	1107066	2
2	1	0	28-r002	1097978	628834	544200	544200	544200	1097978	6
3	1	0	29-r001	1952450	689708	1397469	1397469	1397469	1952450	2
4	1	0	29-r002	1939225	683046	1398572	1398572	1398572	1939225	6
5	1	0	30-r001	2055740	651042	508343	508343	508343	2055740	5

5 rows × 1941 columns

Detecting Potential Outliers using the IQR method

In [94]:

```
outliers_df = non_bias_dummy_df.copy()
```

Group Ionic strength rows together: for calculation of outliers within each subgroup

In [95]:

```
sorted_data = outliers_df.sort_values('Nutrient solution', inplace=False)
#use the drop paramter to prevent old index system being added as a column
sorted_data.reset_index(drop=True, inplace=True)
```

In [96]:

```
no_outliers_df = removeOutliers(sorted_data.columns[3:], sorted_data, 'Nutrient solution',
```

In [97]:

```
##confirm that outliers have been removed / the dataset has been modified
sorted_data.equals(no_outliers_df)
```

Out[97]:

```
False
```

In [98]:

```
#create a list of all chemical compound columns in dataframe
col_names = list(no_outliers_df)
#remove the first three column names (Genotype, ionic strength and CPD ID - as they are not
col_names = col_names[3:]

imputed_df = imputedValues(col_names, no_outliers_df, 'Nutrient solution')
```

In [99]:

```
#confirm that Iterative Imputer has imputed all missing values
print(imputed_df.isnull().values.any())
```

```
False
```

3. Analysis

3. Analysis Technique 1: Kruskal-Wallis Test

In [100]:

```
#retrieve the column names that contain the 5 key metabolic markers
metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

metabol_marker_col_names = []

for col in imputed_df.columns.values:
    for marker in metabol_markers:
        if marker in col:
            metabol_marker_col_names.append(col)
```

In [101]:

```
ns_type_keys = np.array(imputed_df['Nutrient solution'])
kruskal_wallis_results_ns_type = kruskalWallis(metabol_marker_col_names, ns_type_keys, 0.05
print('Metabolic markers correlated with type of nutrient solution:')
kruskal_wallis_results_ns_type
```

```
Metabolic markers correlated with ionic strength:
Metabolic markers correlated with type of nutrient solution:
```

Out[101]:

```
['α-D-glucose 6-phosphate',
 'α-methyl-tyrosine',
 'aldehydo-D-glucose',
 'keto-D-fructose',
 'N,N-dihydroxy-L-valine',
 'O-acetyl-L-serine',
 '1-O-(4-coumaroyl)-β-D-glucose',
 '1-O-malyl-β-D-glucose',
 '1-O-vanilloyl-β-D-glucose',
 '1-16:0-2-18:1-diacylglycerol-trimethylhomoserine',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 '2-deoxyglucose 6-phosphate',
 'L-glucose',
 'L-glutamine',
 'L-serine',
```

3. Analysis Technique 2: Kendall's Tau Correlation Matrix

In [102]:

```
#retrieve the column names that contain the 5 key metabolic markers in plants
metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

col_names = []
for x in imputed_df.columns.values[3:]:
    for marker in metabol_markers:
        if marker in x:
            col_names.append(x)
```

In [103]:

```
corr_prep_df = pd.concat([imputed_df.iloc[:, :3], imputed_df[col_names]], axis="columns")
```

In [104]:

```
correlation_df = corr_prep_df.corr(method='kendall').iloc[:, :2]
```

In [105]:

```
#gather a correlation matrix for the key metabolic markers (included in this dataset)
metabol_corr_df_ns_type = correlation_df.loc[col_names]
metabol_corr_df_ns_type
```

Out[105]:

	Genotype	Nutrient solution
(-)-jasmonoyl-L-valine	0.305690	0.015347
(indol-3-yl)acetyl-L-glutamine	-0.548920	0.214687
(indol-3-yl)acetyl-L-tyrosine	-0.460750	0.287756
α-D-glucose 6-phosphate	-0.048733	0.679104
α-methyl-tyrosine	-0.080000	-0.323316
aldehydo-D-glucose	-0.292632	0.007680
keto-D-fructose	-0.292632	0.007680
N,N-dihydroxy-L-valine	-0.057594	-0.694451
N-hydroxy-L-valine	-0.177071	-0.176350
N,N-dihydroxy-L-tyrosine	-0.248096	0.184164
N<sup>5</sup>-methyl-L-glutamine	0.048694	0.034503
O-acetyl-L-serine	0.101816	-0.463877
1-O-(4-coumaroyl)-β-D-glucose	0.301260	-0.145796
1-O-feruloyl-β-D-glucose	0.247899	0.107344
1-O-malyl-β-D-glucose	0.239426	-0.314863
1-O-vanilloyl-β-D-glucose	-0.128376	-0.141847
1-16:0-2-18:1-diacylglycerol-trimethylhomoserine	-0.274460	0.310530
1-16:0-2-18:2-diacylglycerol-trimethylhomoserine	0.469238	-0.310530
1-16:0-2-18:3-diacylglycerol-trimethylhomoserine	0.309874	0.138013
1-16:0-2-18:4-diacylglycerol-trimethylhomoserine	-0.146083	0.153348
2-deoxyglucose 6-phosphate	-0.168217	-0.341199
6-O-galloylglucose	-0.664016	0.172516
CDP-α-D-glucose	0.159364	-0.069007
D-valine	0.717137	-0.134179
dTDP-α-D-glucose	-0.048733	0.141960
L-cycloserine	0.655162	0.218521
L-glucose	-0.292632	0.007680
L-glutamine	0.004434	-0.026879
L-homoserine	-0.301021	0.023002
L-norvaline	0.717137	-0.134179
L-serine	0.035414	-0.414039
L-tyrosine	0.637455	-0.203186
L-valine	0.717137	-0.134179

	Genotype	Nutrient solution
O-phospho-L-homoserine	-0.225765	-0.072840
O-phospho-L-tyrosine	0.013280	-0.325864
oxindole-3-acetyl-L-valine	0.070941	0.138233
salicylate β-D-glucose ester	0.181498	-0.141847
UDP-α-D-glucose	0.239046	-0.049838

Exploration of optimal Nutrient Solution Type

In [106]:

```
#function meanByCategory requires string keys for each category
def map_nutrient_category(x):
    if x == 0:
        return 'SCa'
    elif x == 1:
        return 'SMg'
    else:
        return 'SK'
```

In [107]:

```
nutrient_type_keys = map(map_nutrient_category, ns_type_keys)
nutrient_type_keys = list(nutrient_type_keys)
nutrient_type_keys= np.array(nutrient_type_keys)
```

In [108]:

```
nutrient_type_keys = np.array(imputed_df['Nutrient solution'])
mean_by_compounds = meanByCategory(imputed_df ,metabol_marker_col_names, nutrient_type_keys
```

In [109]:

```
tallyHighestMeans(mean_by_compounds, ['0','1','2'], 'Nutrient solution Type')
```

```
Number of compounds with highest mean at 0 Nutrient solution Type: 17
Number of compounds with highest mean at 1 Nutrient solution Type: 9
Number of compounds with highest mean at 2 Nutrient solution Type: 12
```

'SCa, SMg, or SK, corresponding to a macrocations' ratio of 0.16 K/0.68 Ca/0.16 Mg, 0.16 K/0.16 Ca/0.68 Mg, and 0.68 K/0.16 Ca/0.16 Mg, respectively.' 0 = 'SCa' 1 = 'SMg' 2 = 'SK'

The results suggest that a Nutrient Solution type SCa optimises the most metabolic markers - this may result in an increased crop yield.

Comparison of Kendall's Tau Results for Each Dataset

In [110]:

```python
#compile results for comparison of pre-harvest factors
correlation_results = metabol_corr_df.copy()
correlation_results['Light Intensity'] = metabol_corr_df_light['Light Intensity']
correlation_results['Nutrient solution'] = metabol_corr_df_ns_type['Nutrient solution']
correlation_results
```

Out[110]:

	Genotype	Ionic strenght	Light Intensity	Nutrient solution
(-)-jasmonoyl-L-valine	-0.243666	0.164980	0.160514	0.015347
(indol-3-yl)acetyl-L-glutamine	-0.372144	-0.103592	-0.310997	0.214687
α-D-glucose 6-phosphate	-0.252326	-0.187851	-0.140450	0.679104
aldehydo-D-glucose	0.235180	0.438086	NaN	0.007680
keto-D-fructose	0.235180	0.438086	NaN	0.007680
N,N-dihydroxy-L-valine	0.318981	0.537144	-0.241209	-0.694451
N-hydroxy-L-valine	0.216912	-0.356534	NaN	-0.176350
O-acetyl-L-serine	0.341132	0.049878	0.020064	-0.463877
1-(2-oxindole-3-acetyl)-β-D-glucose	-0.199363	-0.099755	NaN	NaN
1-O-(4-coumaroyl)-β-D-glucose	-0.057594	0.422042	0.482418	-0.145796
1-O-caffeoyl-β-D-glucose	-0.017707	0.034503	NaN	NaN
1-O-feruloyl-β-D-glucose	-0.225765	0.053672	-0.281924	0.107344
1-O-malyl-β-D-glucose	-0.088676	0.049917	NaN	-0.314863
1-O-sinapoyl-β-D-glucose	0.620734	0.138233	-0.272352	NaN
1-O-vanilloyl-β-D-glucose	-0.301260	-0.268572	-0.140450	-0.141847
1-16:0-2-18:1-diacylglycerol-trimethylhomoserine	-0.026561	0.180184	NaN	0.310530
1-16:0-2-18:2-diacylglycerol-trimethylhomoserine	0.039873	-0.349144	NaN	-0.310530
1-16:0-2-18:3-diacylglycerol-trimethylhomoserine	0.309874	-0.279860	0.291461	0.138013
1-16:0-2-18:4-diacylglycerol-trimethylhomoserine	0.637455	-0.011501	-0.200643	0.153348
2-O-digalloyl-1,3,4,6-tetra-O-β-D-galloylglucose	-0.602041	0.210853	-0.401286	NaN
2-deoxyglucose 6-phosphate	-0.230375	-0.314613	-0.250804	-0.341199
3-O-digalloyl-1,2,4,6-tetra-O-β-D-galloylglucose	-0.602041	0.210853	-0.401286	NaN
4-O-digalloyl-1,2,3,6-tetra-O-β-D-galloylglucose	-0.602041	0.210853	-0.401286	NaN
ADP-α-D-glucose	-0.717707	-0.099755	NaN	NaN
CDP-α-D-glucose	0.048733	0.019184	0.250804	-0.069007
D-glutamine	0.035556	0.257883	0.300965	NaN
D-valine	0.646309	0.042171	0.040129	-0.134179

	Genotype	Ionic strenght	Light Intensity	Nutrient solution
dihydrozeatin-7-N-glucose	-0.575480	-0.030670	NaN	NaN
dTDP-α-D-glucose	0.022151	-0.195674	0.240772	0.141960
L-cycloserine	0.411690	0.295195	0.441415	0.218521
L-glucose	0.235180	0.438086	NaN	0.007680
L-glutamine	0.026603	0.238067	0.300965	-0.026879
L-homoserine	-0.292167	0.003834	-0.371190	0.023002
L-norvaline	0.646309	0.042171	0.040129	-0.134179
L-serine	-0.398726	-0.314613	NaN	-0.414039
L-tyrosine	0.186072	0.203347	0.321029	-0.203186
L-valine	0.646309	0.042171	0.040129	-0.134179
O-phospho-L-homoserine	0.079682	-0.003834	0.250804	-0.072840
O-phospho-L-tyrosine	-0.186220	0.180470	-0.160514	-0.325864
oxindole-3-acetyl-L-valine	0.035527	0.580735	-0.461479	0.138233
UDP-α-D-glucose	0.035442	-0.046041	0.351125	-0.049838

Limitation of datasets 2 and 3 (Light Intensity and Nutrient Solution): In the table above NaN values are present where the compound was removed from the dataset as it had more than 10% of the data missing - this may have introduced bias into the data.

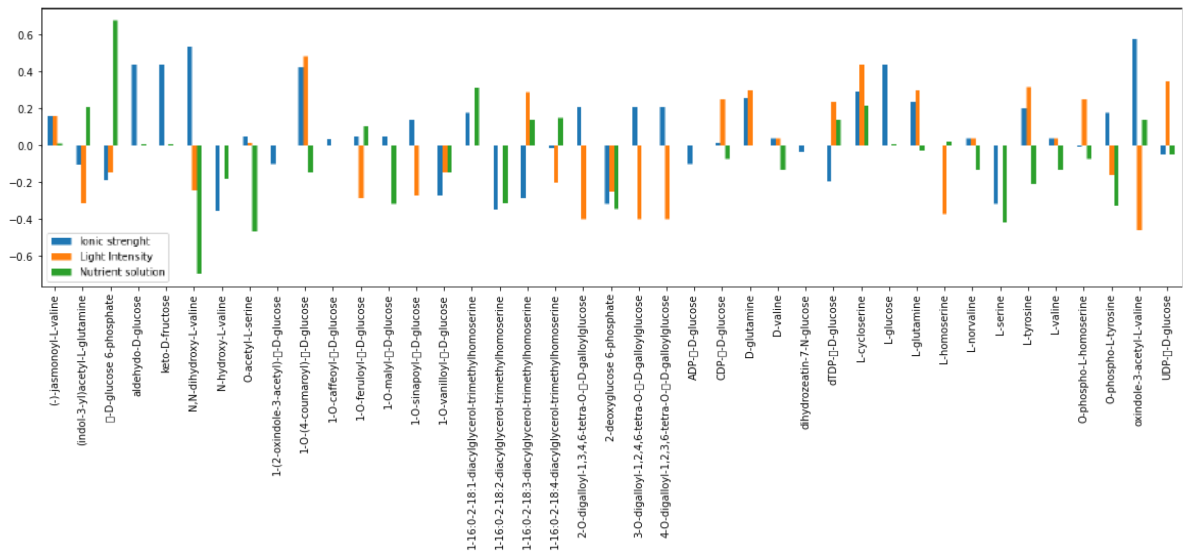
In [111]:

```
#remove genotype to evaluate the impact of the 3 pre-harvest factors without influence of g
correlation_results_no_genotype = correlation_results.drop('Genotype', axis=1, inplace=Fals
```

In [112]:

```
plt.rcParams["figure.figsize"] = (20,5)
bar_plot = correlation_results_no_genotype.plot.bar()
```

```
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120573 missing from current font.
  font.set_text(s, 0, flags=flags)
```



In [113]:

```
ion_stren_count = 0
light_stren_count = 0
ns_type_count = 0
```

In [114]:

```
#if values are negative in the row, find the 'minimum' value (largest negative number)
for index, row in correlation_results_no_genotype.iterrows():
    if all(x <= 0 for x in row.values):
        values = row.values
        #identify index of min value and update the appropriate count variable (see cell ab
        minIndexArr = np.where(values == np.amin(values))
        minIndex = minIndexArr[0][0]
        if minIndex == 0:
            ion_stren_count+=1
        elif minIndex == 1:
            light_stren_count +=1
        elif minIndex== 2:
            ns_type_count +=1
```

In [115]:

```
maxValueIndex = correlation_results_no_genotype.idxmax(axis = 1)

for index, value in maxValueIndex.items():
    if index != '1-O-vanilloyl-β-D-glucose' or index != '2-deoxyglucose 6-phosphate':
        if value == 'Light Intensity':
            light_stren_count +=1
        elif value == 'Ionic strenght':
            ion_stren_count+=1
        elif value == 'Nutrient solution':
            ns_type_count+=1
```

In [116]:

```
print(f"Number of compounds most strongly correlated with ionic strength: {ion_stren_count}
print(f"Number of compounds most strongly correlated with light strength: {light_stren_coun
print(f"Number of compounds most strongly correlated with nutrient solution type: {ns_type_
```

```
Number of compounds most strongly correlated with ionic strength: 22
Number of compounds most strongly correlated with light strength: 12
Number of compounds most strongly correlated with nutrient solution type: 9
```

Based on this analysis, ionic strength seems to have the strongest correlation on the most metabolic markers

Conclusion and Critical Evaluation

Analysis of Kruskal-Wallis results found that changes in ionic strength, light strength and macrocation ratio resulted in statistically different metabolic compound levels;

Compiling and comparing the Kendall’s Tau results for each pre-harvest factor identified which pre-harvest factors impact growth the most as follows:

- Ionic Strength (Electrical Conductivity) of nutrient solution as a dominant factor: 51 % of metabolic markers recorded
- Light Intensity as a dominant factor: 28%
- Ratio of macrocations (K, Ca, and Mg)as a dominant factor : 21%

Cross-validation of Kruskal-Wallis and Kendall’s Tau results for each pre-harvest factor confirmed a correlation between metabolic marker levels.
The results of this investigation indicate that the majority of metabolic markers achieved peak mean value where Hydroponic Crops were fed a Nutrient Solution at full ionic strength (1.5 dsm-1). Ionic strength seems to have the most impact on metabolic marker levels – higher metabolic marker levels lead to an increase in crop yield. The real-world application for this is identifying the optimum Ionic Nutrient Level for maximized growth in Hydroponic Food Production Centres.

Critical Evaluation:
The datasets explored in this analysis use metabolic marker levels as the dependent variable – I have used these metabolic markers to assess growth rate (higher metabolic marker levels may indicate a period of rapid/faster growth). I think it would have been useful if the dataset also measured variables such as dry mass (mass after all water has been removed) to directly indicate growth outcomes as a result of different pre-harvest factors.
The ionic strength dataset tested 3 intervals (1.5 ± 0.1, 0.75 ± 0.1, and 0.5 ± 0.1 dS m−1) ; ionic strengths higher than 1.5 may have the potential to further increase metabolic marker levels.
I think added benefit could be gained by extending the use of Python libraries such as FairLens that can be used to identify bias and generate metrics

Areas for further expansion:
I would be curious to further explore and understand the relationship between ionic strength and metabolic marker levels ; eg deriving an equation through regression. It may be useful to investigate whether genotype impacts metabolic marker levels ; identify if eg the RS genotype has naturally lower levels of a certain compound – understanding this relationship may help to further understand variations in the dataset.
The results of the Kruskal Wallis test indicate that changes in each pre-harvest factor result in statistically different metabolic marker levels – a limitation of the Kruskal Wallis test is that it does not identify between which intervals the statistical significance exists; to identify this a post-hoc test is done (such as Dunn’s test) – this could be used to cross validate the findings on peak mean values.

Ethical Statement

Description of where the data has come from e.g. open or proprietary, licensing and wider considerations around provenance. This dataset was produced through a collaboration of the following institutions: Department of Agricultural Sciences, University of Naples Federico II, 80055 Portici, Italy; Department for Sustainable Food Process, Research Centre for Nutrigenomics and Proteomics, University Cattolica del Sacro Cuore Research Centre for Genomics and Bioinformatics (CREA-GB)

The dataset was found on the MDPI website; an open access database publishing website; MDPI states the following 'Journals published by MDPI are fully open access: research articles, reviews or any other content on this platform is available to everyone free of charge. To be able to provide open access journals, we finance publication through article processing charges (APC); these are usually covered by the authors' institutes or research funding bodies.' Source: https://www.mdpi.com/about (https://www.mdpi.com/about)

The MDPI Publication Ethics statement states that MDPI is a member of the Committee on Publication Ethics (COPE) and that 'DPI takes the responsibility to enforce a rigorous peer-review together with strict ethical policies and standards to ensure to add high quality scientific works to the field of scholarly publication.' Source: https://www.mdpi.com/about (https://www.mdpi.com/about)

Considerations about usage/reusage of data e.g. does the analysis have the potential to create new forms of intellectual property? How is attribution given? The following statements are attached to the Datasets : 'sharing the data will allow to make independent discoveries that, for instance, may stem from the meta-analysis of the different experimental factors, as well as from pre-testing assumptions on specific metabolic pathways or biochemical classes [20,21].' (source: accompanying pdf) These datasets are licenced under creative commons; the following actions are permitted under the creative commons license: 'Share — copy and redistribute the material in any medium or format Adapt — remix, transform, and build upon the material for any purpose, even commercially.'

The creative commons licence states that 'Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.' Source: https://creativecommons.org/licenses/by/4.0/ (https://creativecommons.org/licenses/by/4.0/)

Consideration around implications of utilising data for purpose (e.g. is there power to discriminate? Could research summaries produce dangerous or harmful assumptions?) Factors not considered by this data (such as compound levels that are safe/healthy for human consumption) may lead to potentially harmful conclusions. Incorrect conclusions may lead to an improper/incorrect use of resources (power for light intensity, larger volume of compounds for higher ionic strength), which may result in lower / poorer crop yield.

Considerations of the data processing pipeline. Is the data readily accessible in your notebook? Anonymised? Can it clearly be identified what has been done with the data and that there is no potential for personally identifiable distinctions to be made? 'Recital 26 defines anonymous information, as '… information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable'.' Source: https://www.ucl.ac.uk/data-protection/guidance-staff-students-and-researchers/practical-data-protection-guidance-notices/anonymisation-and (https://www.ucl.ac.uk/data-protection/guidance-staff-students-and-researchers/practical-data-protection-guidance-notices/anonymisation-and) According to this definition, the 3 datasets used in this exploration are anonymous as they do not related to an identifiable natural person.

Any potential biases of the dataset have been considered (e.g. where 80% of the dataset falls into one demographic and 20% for another.) In the PDF accompanying this dataset, the authors declare no conflict of interest – they also state this the research received no external funding. The datasets consider 2 intervals if light intensity (photosynthetic photon flux density (PPFD) of 420 μmolm and 210 μmolm) The 3 datasets analysed explore the effects of 3 ionic strengths: 1.5 ± 0.1, 0.75 ± 0.1, and 0.5 ± 0.1 dS m−1 - it may be that an ionic strength higher than 1.5 or lower than 0.5 may result in optimal crop yield - this may be a potential bias of the dataset that could narrow analysis results. The FairLens library identified an even distribution of data across genotype and ionic strength intervals

Appendix:

Exploration of Ionic Strength on Number of Leaves

Web scraping of Comparison Dataset

In [117]:

```
import requests
import bs4
```

Web scrape the data from the following source: 'Effect of Nutrient Solution Concentration on the Growth of Hydroponic Sweetpotato' https://www.mdpi.com/2073-4395/10/11/1708/htm#:~:text=Because%20the%20nutrient%20solution%20is,to%20growth%20inhibition%20%5E (https://www.mdpi.com/2073-4395/10/11/1708/htm#:~:text=Because%20the%20nutrient%20solution%20is,to%20growth%20inhibition%20%5E

In [118]:

```
link = "https://www.mdpi.com/2073-4395/10/11/1708/htm#:~:text=Because%20the%20nutrient%20so
#implement exception handling for potential errors: URL Exception and HTTP exception
try:
    result = requests.get(link)
except URLError as e:
  print(e)
except HTTPError as e:
  print(e)
```

In [119]:

```
soup = bs4.BeautifulSoup(result.text, "lxml")
```

In [120]:

```
table = soup.find_all('table')
```

In [121]:

```
bio_mass_data = pd.read_html(str(table))[0]
```

In [122]:

```
bio_mass_data
```

Out[122]:

	Treatment	Number of Total Leaves	Abscised Leaf Ratio (%)
0	LL	277 c	63.6 ab
1	LH	545 ab	58.9 c
2	HL	417 bc	72.9 a
3	HH	761 a	66.9 ab

Treatment Key Column:
(1) LL, plants were grown in low EC nutrient solution throughout the cultivation period;
(2) LH, plants were grown in low EC nutrient solution until the end of the first half of the cultivation period, and then transferred to high EC nutrient solution and maintained until the end of the cultivation period;
(3) HL, plants grown in high EC nutrient solution were transferred to low EC nutrient solution at the end of the first half of the cultivation period and maintained thereafter;
(4) HH, plants were grown in high EC nutrient solution throughout the cultivation period.

In [123]:

```
for col_name in ['Number of Total Leaves', 'Abscised Leaf Ratio (%)']:
    bio_mass_data[col_name] = bio_mass_data[col_name].str.slice(0,4)
```

In [124]:

```
bio_mass_data
```

Out[124]:

	Treatment	Number of Total Leaves	Abscised Leaf Ratio (%)
0	LL	277	63.6
1	LH	545	58.9
2	HL	417	72.9
3	HH	761	66.9

In [125]:

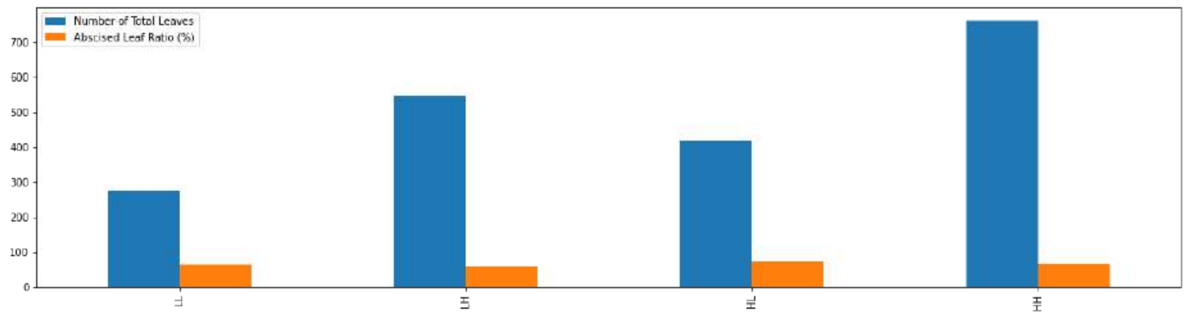
```
new_row_index = bio_mass_data['Treatment'].to_numpy()
bio_mass_data.index = new_row_index
bio_mass_data.drop('Treatment', axis=1, inplace=True)
bio_mass_data
```

Out[125]:

	Number of Total Leaves	Abscised Leaf Ratio (%)
LL	277	63.6
LH	545	58.9
HL	417	72.9
HH	761	66.9

In [126]:

```
bio_mass_data = bio_mass_data.astype({"Number of Total Leaves":"float",
                              "Abscised Leaf Ratio (%)":"float", })
bar_plot = bio_mass_data.plot.bar()
```



Analysing the figure above, this datasets seems to confirm the conclusion that higher ionic strength contributes to higher crop yield; exposing a plant to high ionic strength in both growth periods (HH) has resulted in the largest increase in number of leaves.

In [ ]:

```

```

Analysis Technique 3: Linear Regression

Implement Dummy variables in preparation for regression analysis:

In [127]:

```python
#gather the original categorical variables:
categorical_df = non_bias_df[['Genotype','Ionic strenght']].copy()
categorical_df.head()
```

Out[127]:

	Genotype	Ionic strenght
1	GS	FS
2	GS	FS
3	GS	FS
4	GS	FS
5	GS	FS

In [128]:

```python
#If there are N categories, create N-1 dummy variables, otherwise multicollinearity is intr
dummy_df = pd.get_dummies(categorical_df, drop_first=True)
```

In [129]:

```python
concat_prep = cleaned_data.iloc[: , 3:]
#update to match index of 2 dataframes
concat_prep.index +=1
concatenated = pd.concat([dummy_df, concat_prep], axis="columns")
concatenated.head()
```

Out[129]:

	Genotype_RS	Ionic strenght_HS	Ionic strenght_QS	(-)-(4S)-α-terpineol	(-)-endo-fenchol	(-)-5'-demethylyatein	(-)-9β-7,15-
1	0	0	0	2.525843e+06	2.525843e+06	1067590.0	4.247
2	0	0	0	1.433160e+06	1.433160e+06	1035707.0	1.326
3	0	0	0	2.743781e+06	2.743781e+06	1093665.0	1.326
4	0	0	0	2.743781e+06	2.743781e+06	1128519.0	5.970
5	0	0	0	1.468904e+06	1.468904e+06	1193380.0	5.981

5 rows × 2125 columns

In [130]:

```
dummy_df = concatenated.copy()
```

In [ ]:

```

```

Scaling dependent variable for interpretability of the model ; scaling all dependent variables may also facilitate comparison of regression coefficients

Scaling variables: 'A target variable with a large spread of values, in turn, may result in large error gradient values causing weight values to change dramatically, making the learning process unstable.

Scaling input and output variables is a critical step in using neural network models.'

Source: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/ (https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/)

As the input variables are categorical (dummy variables), I have not standardised them

In [131]:

```
import pandas as pd
from sklearn import preprocessing

scale_prep_df = dummy_df.copy().iloc[:, 3:]
col_names = scale_prep_df.columns.values

x = scale_prep_df.values
#Transform features by scaling each feature to a given range. (0-1 to keep data intuitive a
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df_targets_scaled = pd.DataFrame(x_scaled)

df_targets_scaled.columns = col_names
#make index same format as original data frame (to prepare for concatenation of tables)
df_targets_scaled.index += 1
df_targets_scaled.head()
```

Out[131]:

	(-)-(4S)-α-terpineol	(-)-endo-fenchol	(-)-5'-demethylyatein	(-)-9β-pimara-7,15-dien-19-ol	(-)-9β-pimara-7,15-diene	(-)-borneol	(-)-bursehernin	(-)-carvone
1	0.533550	0.533550	0.400015	0.275644	0.222251	0.533550	0.339418	0.155345
2	0.040241	0.040241	0.385873	0.060701	0.526990	0.040241	0.339418	1.000000
3	0.631942	0.631942	0.411581	0.060701	0.526990	0.631942	0.000000	0.952923
4	0.631942	0.631942	0.427041	0.007056	0.711765	0.631942	0.783383	0.594300
5	0.056378	0.056378	0.455812	0.007135	0.722325	0.056378	0.630256	0.615362

5 rows × 2122 columns

In [132]:

```
#reassemble scaled dataset
scaled_df = pd.concat([dummy_df.iloc[:,:3], df_targets_scaled], axis="columns")
scaled_df.head()
```

Out[132]:

	Genotype_RS	Ionic strenght_HS	Ionic strenght_QS	(-)-(4S)-α-terpineol	(-)-endo-fenchol	(-)-5'-demethylyatein	(-)-9β-pimara-7,15-dien-19-ol	(- pin c
1	0	0	0	0.533550	0.533550	0.400015	0.275644	0.22
2	0	0	0	0.040241	0.040241	0.385873	0.060701	0.52
3	0	0	0	0.631942	0.631942	0.411581	0.060701	0.52
4	0	0	0	0.631942	0.631942	0.427041	0.007056	0.71
5	0	0	0	0.056378	0.056378	0.455812	0.007135	0.72

5 rows × 2125 columns

In [133]:

```
col_names = list(scaled_df)

compound_names = col_names[3:]

affected_compounds = []

independent_var_df = scaled_df.iloc[: , :3]

#print(linearRegression(normalised_df, 'L-glutamine', independent_var_df))

#I originally implemented the following code to create a list of compounds that were affect
#of the nutrient solution - however the for loop was taking a relatively long time to execu
#as the for loop is fitting linear regressions to over 2000 compounds
# for compound in compound_names:
#     #apply linear regression to each compound in the dataset
#     #build a list of compounds that are affected by ionic strength of compound

#     if linearRegression(scaled_df, compound, independent_var_df) != None:
#         affected_compounds.append(linearRegression(scaled_df, compound, independent_var_d
```

As the approach above appears to be to computationally expensive - I refined my search down as to whether the 5 key metabolic markers for plants were affected by changes in ionic strength: the 5 key metabolic markers are: (tyrosine, threonine, valine, serine, glutamine, glucose and fructose) Source: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5025497/#:~:text=Five%20amino%20acids%20 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5025497/#:~:text=Five%20amino%20acids%20) (tyrosine%2C%20threonine,%2C%20respectively%20(Steinfath%20et%20al.

Retrieving column names relevant to metabolic markers:

In [134]:

```
#retrieve the column names that contain the 5 key metabolic markers
metabol_markers = ["tyrosine", "threonine", "valine", "serine", "glutamine", "glucose","fru

metabol_marker_col_names = []

for col in scaled_df.columns.values:
    for marker in metabol_markers:
        if marker in col:
            metabol_marker_col_names.append(col)


metabol_marker_col_names
```

Out[134]:

```
['(-)-jasmonoyl-L-valine',
 '(indol-3-yl)acetyl-L-glutamine',
 'α-D-glucose 6-phosphate',
 'aldehydo-D-glucose',
 'keto-D-fructose',
 'N,N-dihydroxy-L-valine',
 'N-hydroxy-L-valine',
 'O-acetyl-L-serine',
 '1-(2-oxindole-3-acetyl)-β-D-glucose',
 '1-O-(4-coumaroyl)-β-D-glucose',
 '1-O-caffeoyl-β-D-glucose',
 '1-O-feruloyl-β-D-glucose',
 '1-O-malyl-β-D-glucose',
 '1-O-sinapoyl-β-D-glucose',
 '1-O-vanilloyl-β-D-glucose',
 '1-16:0-2-18:1-diacylglycerol-trimethylhomoserine',
 '1-16:0-2-18:2-diacylglycerol-trimethylhomoserine',
 '1-16:0-2-18:3-diacylglycerol-trimethylhomoserine',
 '1-16:0-2-18:4-diacylglycerol-trimethylhomoserine',
 '2-O-digalloyl-1,3,4,6-tetra-O-β-D-galloylglucose',
 '2-deoxyglucose 6-phosphate',
 '3-O-digalloyl-1,2,4,6-tetra-O-β-D-galloylglucose',
 '4-O-digalloyl-1,2,3,6-tetra-O-β-D-galloylglucose',
 'ADP-α-D-glucose',
 'CDP-α-D-glucose',
 'D-glutamine',
 'D-valine',
 'dihydrozeatin-7-N-glucose',
 'dTDP-α-D-glucose',
 'L-cycloserine',
 'L-glucose',
 'L-glutamine',
 'L-homoserine',
 'L-norvaline',
 'L-serine',
 'L-tyrosine',
 'L-valine',
 'O-phospho-L-homoserine',
 'O-phospho-L-tyrosine',
 'oxindole-3-acetyl-L-valine',
 'UDP-α-D-glucose']
```

Definition of a function to create a linear regression for each of the compounds listed above:

In [135]:

```
#accepts a python dataframe containing genotype, independent variable (such as ionic streng
import scipy.stats as stats
from scipy import stats
#keyError - need to allow for ionic_strength_HS and QS for access
def linearRegression(pandasDataFrame, dependentColName, indepedentVarDataFrame):

    try:

        #create a multiple linear regression
        y = pandasDataFrame[dependentColName]
        x1 = indepedentVarDataFrame.copy()

        x = sm.add_constant(x1)
        #type casting is required as variables are lifted from a dataframe
        #source: https://stackoverflow.com/questions/33833832/building-multi-regression-mod
        multiple_reg_results = sm.OLS(y.astype(float),x.astype(float)).fit()
        multiple_reg_adjusted_rsquared = multiple_reg_results.rsquared_adj

        #create a linear regression (without genotype)
        y2 = pandasDataFrame[dependentColName]
        x2 = pandasDataFrame.drop(['Genotype_RS'], axis=1)

        x = sm.add_constant(x2)
        #type casting is required as variables are lifted from a dataframe
        #source: https://stackoverflow.com/questions/33833832/building-multi-regression-mod
        results = sm.OLS(y2.astype(float),x.astype(float)).fit()
        results_adjusted_rsquared = results.rsquared_adj

        alpha = 0.05
        #penalised for including a variable with no strong explanatory power
        if multiple_reg_adjusted_rsquared < results_adjusted_rsquared:
            #confirm that model residuals are normally distributed
            shapiro_test = stats.shapiro(multiple_reg_results.resid)
            if shapiro_test.pvalue <= alpha:
                if results.pvalues['Ionic strenght_HS'] < alpha or results.pvalues['Ionic s
                    #print(f"Ionic Strength is a significant variable correlated with {depe
                    #determine which coefficient is highest
                    ionic_stren_coeff_QS = round(results.params['Ionic strenght_QS'], 3)
                    ionic_stren_coeff_HS = round(results.params['Ionic strenght_HS'], 3)
                    largest_coeff = ionic_stren_coeff_QS if (ionic_stren_coeff_QS > ionic_s

                    #print('The adjusted R-squared value indicates that Genotype offers no
                    #print('power to the model\n')
                    return { "compound": dependentColName,
                             "ionic_strength": "ionic_stren_coeff_QS" if (ionic_stren_coef
                              "ionic_reg_coefficient": largest_coeff,
                              "coeff_polarity": "positive" if largest_coeff >0 else 'negat
        else:
            #confirm that model residuals are normally distributed
            shapiro_test = stats.shapiro(multiple_reg_results.resid)
            if shapiro_test.pvalue <= alpha:
                if multiple_reg_results.pvalues['Ionic strenght_HS'] < alpha or multiple_re
                    #print(f"Ionic Strength is a significant variable correlated with {depe
                    #determine which coefficient is highest
                    ionic_stren_coeff_QS = round(multiple_reg_results.params['Ionic strengh
                    ionic_stren_coeff_HS = round(multiple_reg_results.params['Ionic strengh
                    largest_coeff = ionic_stren_coeff_QS if (ionic_stren_coeff_QS > ionic_s

                    genotype_coeff = round(multiple_reg_results.params['Genotype_RS'],5)
                    #print('The adjusted R-squared value indicates that Genotype offers sta
                    #print('power to the model\n')

                    if genotype_coeff < alpha:
                        return { "compound": dependentColName,
                             "ionic_strength": "ionic_stren_coeff_QS" if (ionic_stren_coef
                              "ionic_reg_coefficient": largest_coeff,
                               "genotype_coeff": genotype_coeff,
                              "coeff_polarity": "positive" if largest_coeff >0 else 'negat
                    else:
                        return { "compound": dependentColName,
                             "ionic_strength": "ionic_stren_coeff_QS" if (ionic_stren_coef
                              "ionic_reg_coefficient": largest_coeff,
                              "coeff_polarity": "positive" if largest_coeff >0 else 'negat

   except KeyError:
     print("Please confirm you are passing a valid column name to the last 3 parameters")
   except Exception as e:
        print(e)
```

In [136]:

```
affected_markers = []
independent_var_df = scaled_df.iloc[: , :3]

for marker in metabol_marker_col_names:
    #apply linear regression to each metabolic marker
    #build a list of compounds that are affected by ionic strength of compound

    if linearRegression(scaled_df, marker, independent_var_df) != None:
        affected_markers.append(linearRegression(scaled_df, marker, independent_var_df))
```

```
C:\Users\amyti\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:14
2: FutureWarning: In a future version of pandas all arguments of concat ex
cept for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
C:\Users\amyti\anaconda3\lib\site-packages\statsmodels\regression\linear_m
odel.py:1728: RuntimeWarning: divide by zero encountered in true_divide
  return 1 - (np.divide(self.nobs - self.k_constant, self.df_resid)
C:\Users\amyti\anaconda3\lib\site-packages\statsmodels\regression\linear_m
odel.py:1728: RuntimeWarning: invalid value encountered in double_scalars
  return 1 - (np.divide(self.nobs - self.k_constant, self.df_resid)
C:\Users\amyti\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:14
2: FutureWarning: In a future version of pandas all arguments of concat ex
cept for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
C:\Users\amyti\anaconda3\lib\site-packages\statsmodels\regression\linear_m
odel.py:1728: RuntimeWarning: divide by zero encountered in true_divide
  return 1 - (np.divide(self.nobs - self.k_constant, self.df_resid)
C:\Users\amyti\anaconda3\lib\site-packages\statsmodels\regression\linear_m
odel.py:1728: RuntimeWarning: invalid value encountered in double_scalars
```

In [137]:

```
affected_markers
```

Out[137]:

```
[{'compound': 'α-D-glucose 6-phosphate',
  'ionic_strength': 'ionic_stren_HS',
  'ionic_reg_coefficient': -0.265,
  'genotype_coeff': -0.05674,
  'coeff_polarity': 'negative'},
 {'compound': 'oxindole-3-acetyl-L-valine',
  'ionic_strength': 'ionic_stren_coeff_QS',
  'ionic_reg_coefficient': 0.205,
  'coeff_polarity': 'positive'}]
```

In [138]:

```
len(affected_markers)
```

Out[138]:

```
2
```

Polarity of dummy variables: A positive weight shows that the respective category is more expensive than the benchmark
A negative weight shows that the respective category is less expensieve than the benchmark

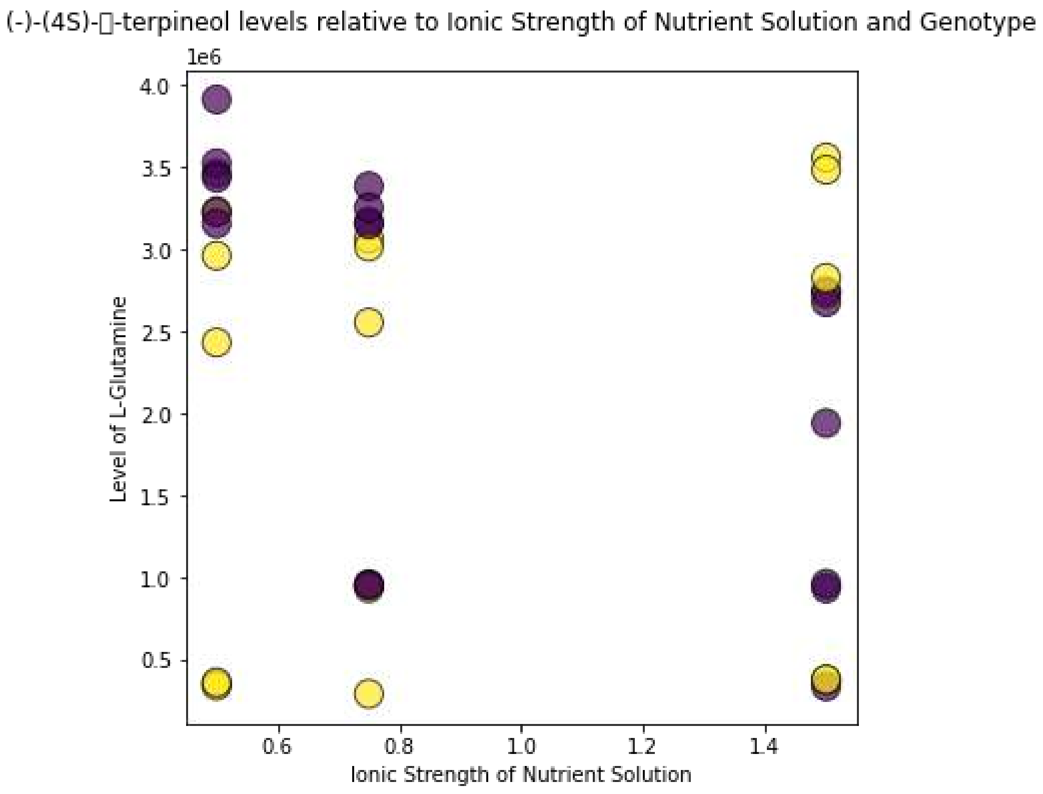
benchmark is the dummy variable that has been dropped Source: https://www.udemy.com/course/the-data-science-course-complete-data-science-bootcamp/learn/lecture/14390492#notes (https://www.udemy.com/course/the-data-science-course-complete-data-science-bootcamp/learn/lecture/14390492#notes)

Visual assessment:

In [139]:

```
x = cleaned_data['Ionic strenght']
y = cleaned_data['𝛼-D-glucose 6-phosphate']
z = cleaned_data['Genotype']

fig = plt.figure(figsize=(6, 6))
plt.scatter(x, y,
            linewidths=1, alpha=.7,
            edgecolor='k',
            s = 200,
            c=z)
plt.title("(-)-(4S)-𝛼-terpineol levels relative to Ionic Strength of Nutrient Solution and
plt.xlabel("Ionic Strength of Nutrient Solution")
plt.ylabel("Level of L-Glutamine")
plt.show()
```

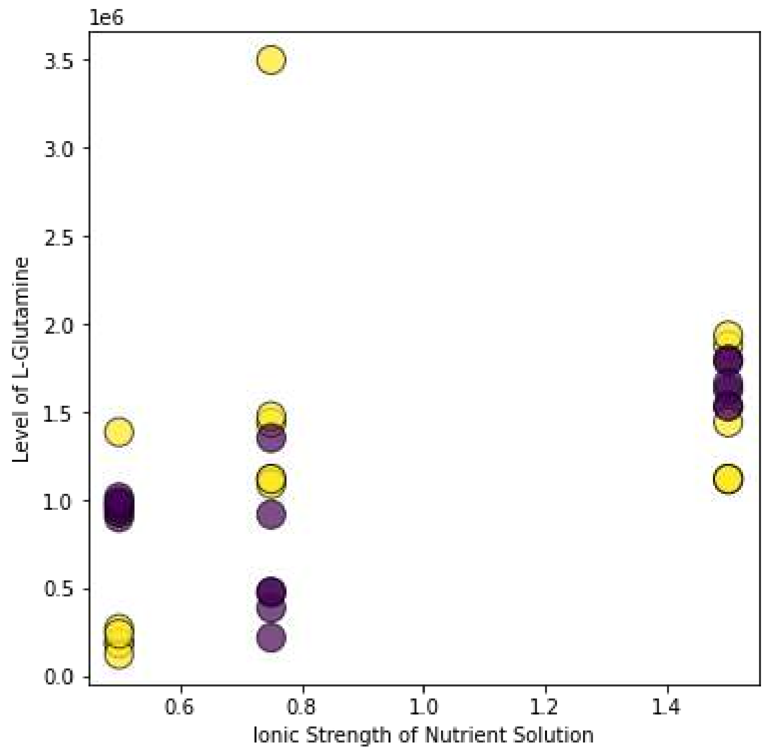
```
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:240: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\Users\amyti\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.p
y:203: RuntimeWarning: Glyph 120572 missing from current font.
  font.set_text(s, 0, flags=flags)
```



In [140]:

```
x = cleaned_data['Ionic strenght']
y = cleaned_data['oxindole-3-acetyl-L-valine']
z = cleaned_data['Genotype']

fig = plt.figure(figsize=(6, 6))

# ax = plt.gca()
# ax.set_ylim([0, 5])
# plt.axis((0,1.5,0,5))

plt.scatter(x, y,
            linewidths=1, alpha=.7,
            edgecolor='k',
            s = 200,
            c=z)
#plt.ylim(-1, 10000000)

plt.title("(-)-(4S)-α-terpineol levels relative to Ionic Strength of Nutrient Solution and
plt.xlabel("Ionic Strength of Nutrient Solution")
plt.ylabel("Level of L-Glutamine")
plt.show()
```

(-)-(4S)-□-terpineol levels relative to Ionic Strength of Nutrient Solution and Genotype



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```