1 | BACKGROUND

1.1 | Context

The CalCOFI data set represents the longest (1949-present) and most complete (more than 50,000 sampling stations) time series of oceanographic and larval fish data in the world. It includes abundance data on the larvae of over 250 species of fish; larval length frequency data and egg abundance data on key commercial species; and oceanographic and plankton data. The physical, chemical, and biological data collected at regular time and space intervals quickly became valuable for documenting climatic cycles in the California Current and a range of biological responses to them. CalCOFI research drew world attention to the biological response to the dramatic Pacific-warming event in 1957-58 and introduced the term “El Niño” into the scientific literature.

CalCOFI conducts quarterly cruises off southern & central California, collecting a suite of hydrographic and biological data on station and underway. Data collected at depths down to 500 m include: temperature, salinity, oxygen, phosphate, silicate, nitrate and nitrite, chlorophyll, transmissometer, PAR, C14 primary productivity, phytoplankton biodiversity, zooplankton biomass, and zooplankton biodiversity.

1.2 | Scope

We would like to know if a connection between Salinity and Temperature exists

2 | REFERENCE LINKS

Main Website: https://calcofi.com/
Bottle-Dataset: https://new.data.calcofi.com/index.php/database/calcofi-database/bottle-field-descriptions
Cast-Dataset: https://new.data.calcofi.com/index.php/database/calcofi-database/cast-table-column-descriptions

3 | EDA (EXPLORATION DATA ANALYSIS)

3.1 | Import Packages and Datasets

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import seaborn as sns
import plotly.express as px
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this ve
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
bot=pd.read_csv('/content/bottle.csv',low_memory=False)#.to_csv('bottlecsv.csv')
cast=pd.read_csv('/content/cast.csv',low_memory=False)#.to_csv('castcsv.csv')
```

immagine.png

```
bot.info()
```

```
 26  PO4q                451786 non-null  float64
 27  SiO3uM              354091 non-null  float64
 28  SiO3qu              510866 non-null  float64
 29  NO2uM               337576 non-null  float64
 30  NO2q                529474 non-null  float64
 31  NO3uM               337403 non-null  float64
 32  NO3q                529933 non-null  float64
 33  NH3uM               64962 non-null   float64
 34  NH3q                808299 non-null  float64
 35  C14As1              14432 non-null   float64
 36  C14A1p              12760 non-null   float64
 37  C14A1q              848605 non-null  float64
 38  C14As2              14414 non-null   float64
 39  C14A2p              12742 non-null   float64
 40  C14A2q              848623 non-null  float64
 41  DarkAs              22649 non-null   float64
 42  DarkAp              20457 non-null   float64
 43  DarkAq              840440 non-null  float64
 44  MeanAs              22650 non-null   float64
 45  MeanAp              20457 non-null   float64
 46  MeanAq              840439 non-null  float64
 47  IncTim              14437 non-null   object
 48  LightP              18651 non-null   float64
 49  R_Depth             864863 non-null  float64
 50  R_TEMP              853900 non-null  float64
 51  R_POTEMP            818816 non-null  float64
 52  R_SALINITY          817509 non-null  float64
 53  R_SIGMA             812007 non-null  float64
 54  R_SVA               812092 non-null  float64
 55  R_DYNHT             818206 non-null  float64
 56  R_O2                696201 non-null  float64
 57  R_O2Sat             666448 non-null  float64
 58  R_SIO3              354099 non-null  float64
 59  R_PO4               413325 non-null  float64
 60  R_NO3               337411 non-null  float64
 61  R_NO2               337584 non-null  float64
 62  R_NH4               64982 non-null   float64
 63  R_CHLA              225276 non-null  float64
 64  R_PHAEO             225275 non-null  float64
 65  R_PRES              864863 non-null  int64
 66  R_SAMP              122006 non-null  float64
 67  DIC1                1999 non-null    float64
 68  DIC2                224 non-null     float64
 69  TA1                 2084 non-null    float64
 70  TA2                 234 non-null     float64
 71  pH2                 10 non-null      float64
 72  pH1                 84 non-null      float64
 73  DIC Quality Comment  55 non-null      object
dtypes: float64(65), int64(5), object(4)
memory usage: 488.3+ MB
```

It has been decided to reduce the dataset to the following features:

```
bottle=bot.iloc[:,[0,4,5,6,7,8]]
bottle
```

	Cst_Cnt	Depthm	T_degC	Salnty	O2ml_L	STheta
0	1	0	10.500	33.4400	NaN	25.64900
1	1	8	10.460	33.4400	NaN	25.65600
2	1	10	10.460	33.4370	NaN	25.65400
3	1	19	10.450	33.4200	NaN	25.64300
4	1	20	10.450	33.4210	NaN	25.64300
...	...	...	...	...	...	...
864858	34404	0	18.744	33.4083	5.805	23.87055
864859	34404	2	18.744	33.4083	5.805	23.87072
864860	34404	5	18.692	33.4150	5.796	23.88911
864861	34404	10	18.161	33.4062	5.816	24.01426
864862	34404	15	17.533	33.3880	5.774	24.15297

864863 rows × 6 columns

```
bottle.info()
```

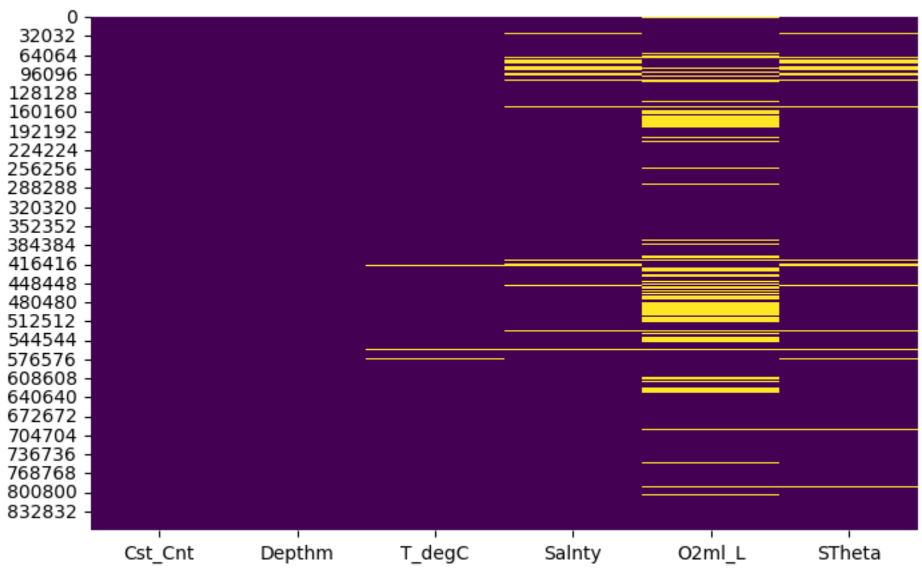
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 864863 entries, 0 to 864862
Data columns (total 6 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   Cst_Cnt  864863 non-null  int64  
 1   Depthm   864863 non-null  int64  
 2   T_degC   853900 non-null  float64
 3   Salnty   817509 non-null  float64
 4   O2ml_L   696201 non-null  float64
 5   STheta   812174 non-null  float64
dtypes: float64(4), int64(2)
memory usage: 39.6 MB
```

3.2 | NAN VALUES ANALYSIS

Nan Values Visualization

```
plt.rcParams["figure.figsize"] = (8,5)
sns.heatmap(bottle.isna(),cbar=False,cmap='viridis')
```

```
<AxesSubplot: >
```



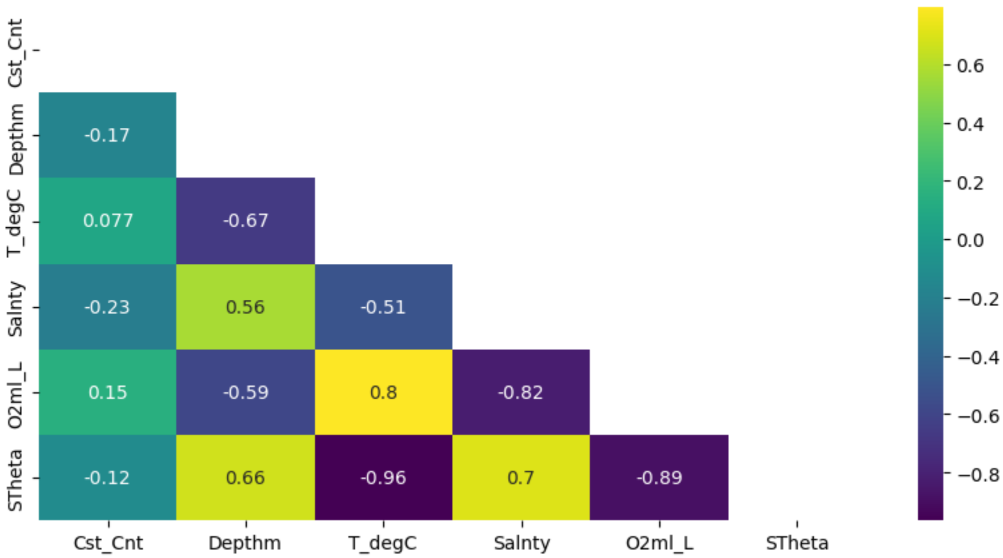
Data cleaning

```
cols=bottle.columns

for colonna in cols:
    bottle=bottle[~bottle[colonna].isnull()]


bottle=bottle.reset_index(drop=True)
plt.rcParams["figure.figsize"] = (8,5)
sns.heatmap(bottle.isna(),cbar=False,cmap='viridis')
```

<AxesSubplot: >

3.3 | CORRELATION MATRIX

```
plt.figure(figsize=(10,5))
mask = np.triu(np.ones_like(bottle.corr(numeric_only = True), dtype=bool))
sns.heatmap(bottle.corr(numeric_only = True), cbar = True, annot = True, cmap="viridis", mask = mask);
```



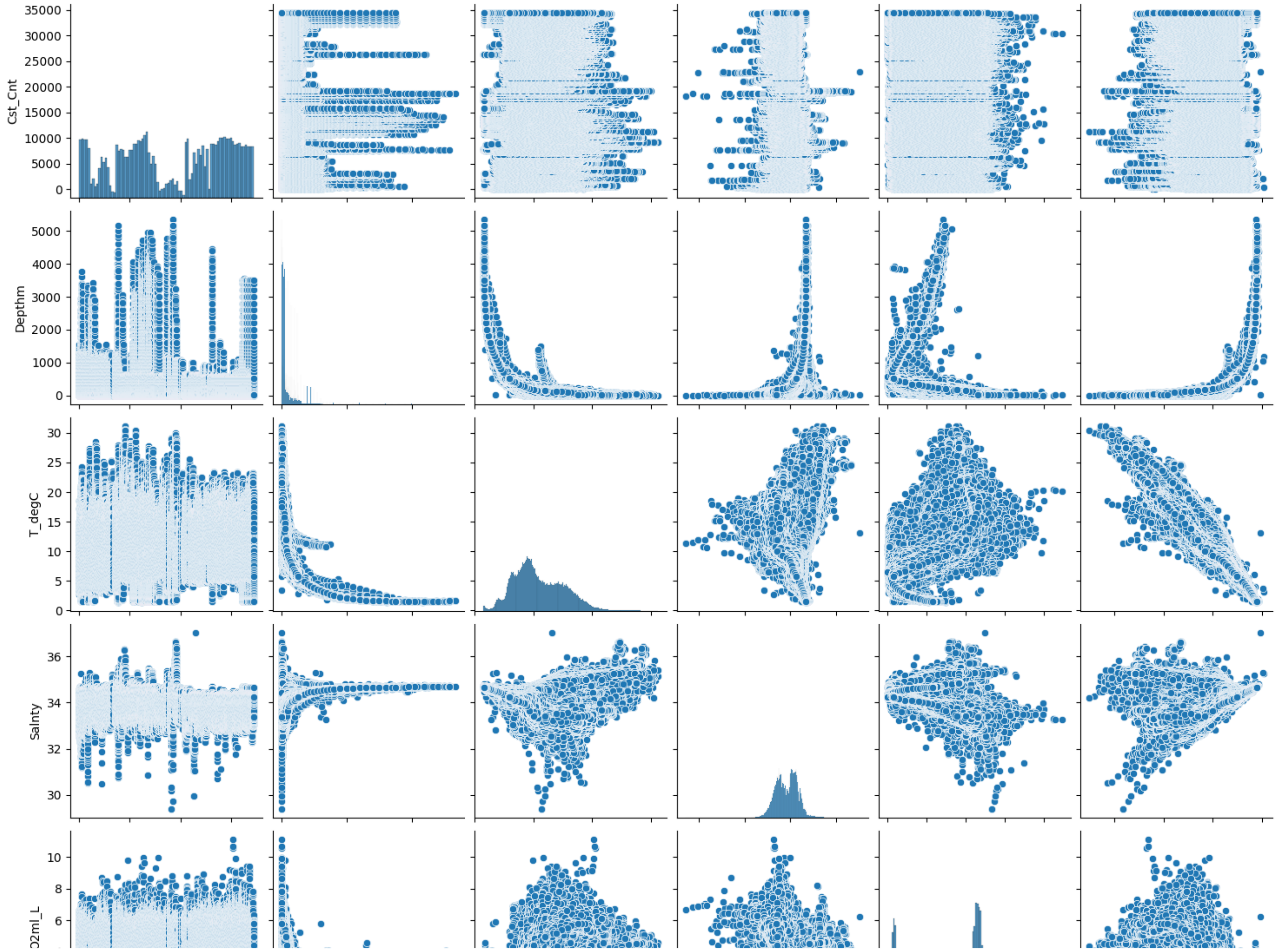
The Pearson's coefficient between Temperature and Salinity is not so high, -0.51

3.4 | FEATURES PLOTS

Plot pairwise relationships in the dataset

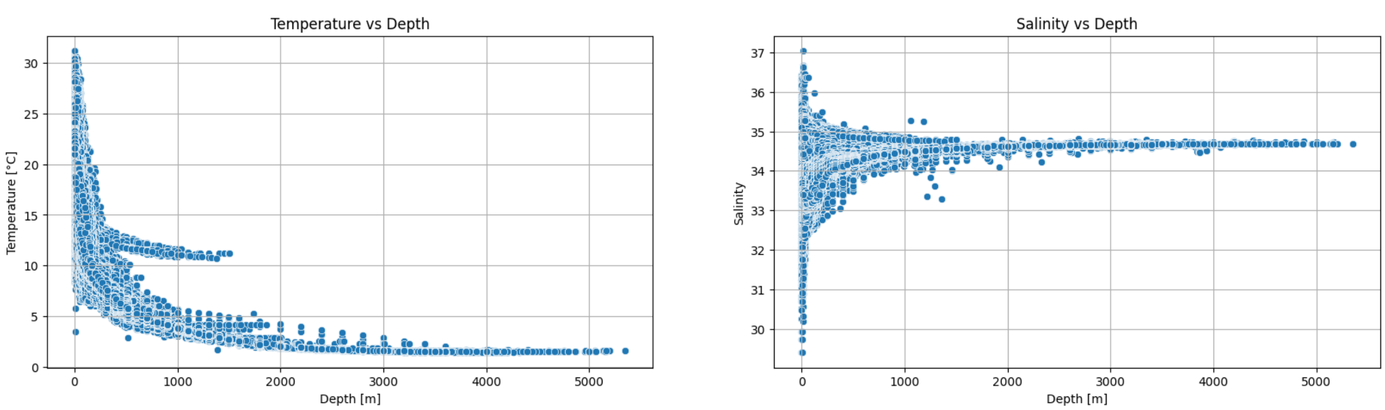
```
sns.pairplot(bottle)
```

<seaborn.axisgrid.PairGrid at 0x7c9465acf7f0>



3.5 | DEPTH - TEMPERATURE - SALINITY


```
fig, ax = plt.subplots(1,2, figsize=(20,5))
g1=sns.scatterplot(data=bottle,x='Depthm',y='T_degC',ax=ax[0])
ax[0].set(xlabel='Depth [m]',ylabel='Temperature [°C]',title='Temperature vs Depth')
ax[0].grid()
g2=sns.scatterplot(data=bottle,x='Depthm',y='Salnty',ax=ax[1])
ax[1].set(xlabel='Depth [m]',ylabel='Salinity',title='Salinity vs Depth')
ax[1].grid()
```



As we can see from the image above temperature and salinity trace a hyperbolic trend as depth changes:

- In the first plot some points in the upper part of the image show a different trend from the rest of the graph. It's an unexpected behavior, because, for instance, at around 1000 m depth the expected temperature is around 5°C, but we also find temperatures between 10°C and 15°C (https://www.windows2universe.org/earth/Water/temp.html)
- In the second plot we can notice that the points converge asymptotically from both high and low salinity values. It means that in some areas of the ocean the salinity encreases with the depth, vice versa in other areas it decreases.

Distribution of Measuring Points on Geograph Datamap

Here we show the measuring points on the map using latitude and longitude informations.

```
df_merge=bottle.merge(cast, how='inner',on='Cst_Cnt')
```

```
fig = px.scatter_mapbox(df_merge,
                        lat="Lat_Dec",
                        lon="Lon_Dec",
                        #hover_name="Address",
                        hover_data=["Depthm"],
                        color="T_degC",
                        #color_continuous_scale=color_scale,
                        #size="Listed",
                        #animation_frame='Year',
                        zoom=3,
                        height=800,
                        width=800)

#fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(
    mapbox_style="white-bg",
    mapbox_layers=[
        {
            "below": 'traces',
            "sourcetype": "raster",
            "sourceattribution": "United States Geological Survey",
            "source": [
                "https://basemap.nationalmap.gov/arcgis/rest/services/USGSImageryOnly/MapServer/tile/{z}/{y}/{x}"
            ]
        }])
fig.show()
```

If we isolate points with salinity values in the range going from 34.85 to 38 we observe our alternative trend highlighted in the Temperature-Depth graph. It came out!

```
fig, ax = plt.subplots(1,2, figsize=(20,5))
bottle_filter=bottle[(bottle['Salnty']>34.85)&(bottle['Salnty']<38)]
g1=sns.scatterplot(data=bottle,x='Depthm',y='T_degC',ax=ax[0])
g1=sns.scatterplot(data=bottle_filter,x='Depthm',y='T_degC',ax=ax[0],legend=None,alpha=0.7)
ax[0].set(xlabel='Depth [m]',ylabel='Temperature [°C]',title='Temperature vs Depth')
ax[0].grid()
g2=sns.scatterplot(data=bottle,x='Depthm',y='Salnty',ax=ax[1])
g2=sns.scatterplot(data=bottle_filter,x='Depthm',y='Salnty',ax=ax[1],alpha=0.7)
ax[1].set(xlabel='Depth [m]',ylabel='Salinity',title='Salinity vs Depth')
ax[1].grid()
```

Distribution of Measuring Points for filtered dataset

Then we show the measuring points from the filtered dataset in the specific salinity range: 34.85 - 38.

```
df_merge_filter=bottle_filter.merge(cast, how='inner',on='Cst_Cnt')

fig = px.scatter_mapbox(df_merge_filter,
                        lat="Lat_Dec",
                        lon="Lon_Dec",
                        #hover_name="Address",
                        hover_data=["Depthm","Salnty","Cst_Cnt"],
                        color="T_degC",
                        #color_continuous_scale=color_scale,
                        #size="Listed",
                        #animation_frame='Year',
                        zoom=5,
                        height=800,
                        width=800)

#fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(
    mapbox_style="white-bg",
    mapbox_layers=[
        {
            "below": 'traces',
            "sourcetype": "raster",
            "sourceattribution": "United States Geological Survey",
            "source": [
                "https://basemap.nationalmap.gov/arcgis/rest/services/USGSImageryOnly/MapServer/tile/{z}/{y}/{x}"
            ]
        }])
fig.show()
```

We note that most points with high salinity values are located in the gulf.

3.6 | SALINITY - TEMPERATURE

```
plt.rcParams['figure.figsize']=10,6
ax=sns.scatterplot(data=bottle,x='Salnty',y='T_degC')
ax.set(xlabel='Salinity',ylabel='Temperature',title='Salinity vs Temperature')
plt.grid()
```

Salinity and Temperature don't seem to have a strong relationship.

We can try to use some additional information using the potential density (https://en.wikipedia.org/wiki/Potential_density). Remind that potential density is not a measured but a calculated quantity from Temperature, Pressure and Salinity.

Thanks to proportional link between chlorine and salinity, the last one (difficult to measure) is calculated through the measurement of chlorine, using an electrical conductivity meter.

The technology used to measure Conductivity, Temperature, and Pressure (related to Depth) at the same time in seawater is called CTD (https://en.wikipedia.org/wiki/CTD_(instrument))

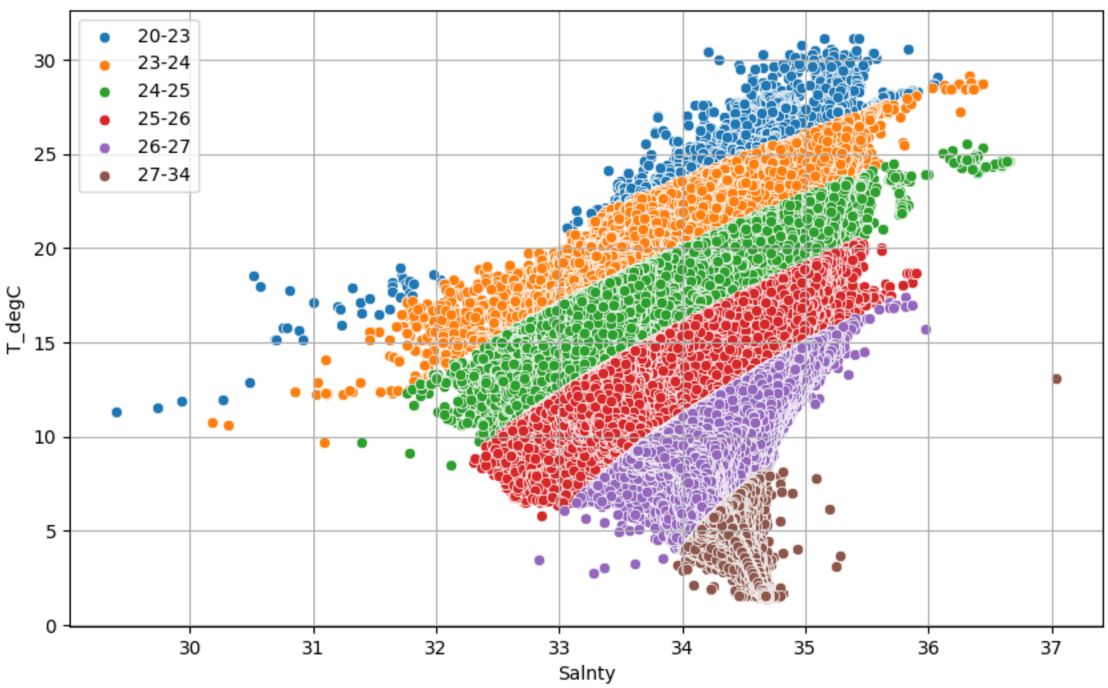
Salinity-Temperature and potential Density plot

As predicted by the theoretical model, Temperature can be expressed as a function of Salinity and Density:

immagine.png

We can try to highlight the density variable on the same graph:

```
theta_step=[20,23,24,25,26,27,34]
plt.rcParams['figure.figsize']=10,6

for theta1, theta2 in zip(theta_step, theta_step[1:]):
    bottle_stheta=bottle[(bottle['STheta']>=theta1) & (bottle['STheta']<theta2)]
    sns.scatterplot(data=bottle_stheta,y='T_degC',x='Salnty',label=str(theta1)+'-'+str(theta2))

plt.grid()
```



It works! But remind, it's not the real density at the specific depth, it's a potential density.

4 | PREDICTIVE MODELS

Split Train - Test

```
### Split
bottle_train, bottle_test = train_test_split(bottle, test_size=0.2, random_state=1234)
```

```
print("bottle_train: ", bottle_train.shape)
print("bottle_test: ", bottle_test.shape,"\n")

#features =['Salnty','STheta']
features =bottle.columns.drop('T_degC').tolist()
target = 'T_degC'
X_train = bottle_train[features].values
y_train = bottle_train[target].values
X_test = bottle_test[features].values
y_test = bottle_test[target].values
```

```
bottle_train:  (529014, 6)
bottle_test:  (132254, 6)
```

4.1 | LINEAR REGRESSION

Temperature from Salinity

```
features1=['Salnty'] #only Salinity feature
X_train1 = bottle_train[features1].values
y_train1 = bottle_train[target].values
X_test1 = bottle_test[features1].values
y_test1 = bottle_test[target].values

# creazione modello regressione lineare
model = LinearRegression()
model.fit(X_train1,y_train1)
y_predict1 = model.predict(X_test1)
```

Linear Regression Error

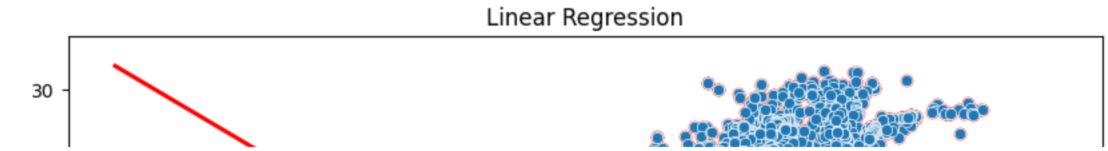
```
#calcolo errore modello di regressione
print(model)
mse_lin=mean_squared_error(y_test1, y_predict1)
RMSE_lin=mse_lin**(1/2.0)

print('RMSE_lin:',RMSE_lin)
```

```
LinearRegression()
RMSE_lin: 3.6326596110432914
```

Linear Regression Plot

```
g2=sns.regplot(x='Salnty',y="T_degC",data=bottle, color="red")
sns.scatterplot(data=bottle,x='Salnty',y='T_degC')
g2.set(xlabel='Salinity', ylabel='Temperature',title='Linear Regression')
```

```
[Text(0.5, 0, 'Salinity'),
 Text(0, 0.5, 'Temperature'),
 Text(0.5, 1.0, 'Linear Regression')]
```



It doesn't seem to exist a clear trend. It's consistently with the -0.5 Pearson's coefficient got from the confusion matrix. The linear regression is inversely proportional, but it doesn't seem a strong correlation. Of course we isolated the feature of Salinity from the others. We could try to use it with other phisical quantities that we've maintained in this work

Multiple Linear Regression

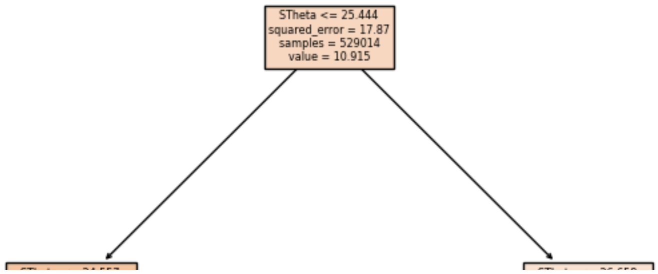
```
# creazione modello regressione lineare
model = LinearRegression()
model.fit(X_train,y_train)
print(model)
y_predict2 = model.predict(X_test)

#calcolo errore modello di regressione
mse_multilin=mean_squared_error(y_test, y_predict2)
RMSE_multilin=mse_multilin**(1/2.0)

print('RMSE_multilin:',RMSE_multilin)
```

```
LinearRegression()
RMSE_multilin: 0.3323601966453793
```

Much better

4.2 | REGRESSION TREE

```
model_tree = DecisionTreeRegressor(random_state=44,max_depth=10)
model_tree.fit(X_train, y_train)
predictions = model_tree.predict(X_test)


plt.figure(figsize=(10,10))
#plot decision tree depth=4
tree.plot_tree(model_tree,feature_names=features,filled=True,max_depth=2);
```



Regression Tree Error

/ \ / \

```
mse_tree = mean_squared_error(y_test, predictions)
RMSE_tree= mse_tree**(1/2)
#print("MSE: ", mse_tree)
print("RMSE_tree: ", RMSE_tree)
```

```
RMSE_tree:  0.18035450438720613
```

4.3 | KNN

/ \ / \ / \ / \

```
k=100
model = KNeighborsRegressor(n_neighbors=k)
print(model)

model.fit(X_train,y_train)

pred_y = model.predict(X_test)
```

```
KNeighborsRegressor(n_neighbors=100)
```

KNN Error

```
print(model)
mse_knn = mean_squared_error(y_test, pred_y)
RMSE_knn = mse_knn**(1/2)
print("RMSE_KNN: ", RMSE_knn)
```

```
KNeighborsRegressor(n_neighbors=100)
RMSE_KNN:  1.3157370402918163
```

KNN alghoritm is worse than multilinear regressor and decision tree regressor.

4.4 | FEATURE IMPORTANCE ANALYSIS

We can get a feature importance analysis through the decision tree model used above

```
for i,feature in enumerate(features):
    print('',feature,'=', model_tree.feature_importances_[i])
```

```
 Cst_Cnt = 8.900949273136677e-06
 Depthm = 0.019813619791390027
 Salnty = 0.06121591882241469
 O2ml_L = 4.917424923116525e-05
 STheta = 0.9189123861876909
```

```
pd.Series(model_tree.feature_importances_[1:5] , index=['Depth','Salinity','Oxygen','Density']).nlargest(4).plot(kind='barh',figsize=(10,4))
```

<AxesSubplot: >

Density (STheta=0.919) and Salinity (Salnty=0.061) have the higher feature importance as shown in the theoretical model

immagine.png

So, we can try to repeat our predictions using only these forementioned two features

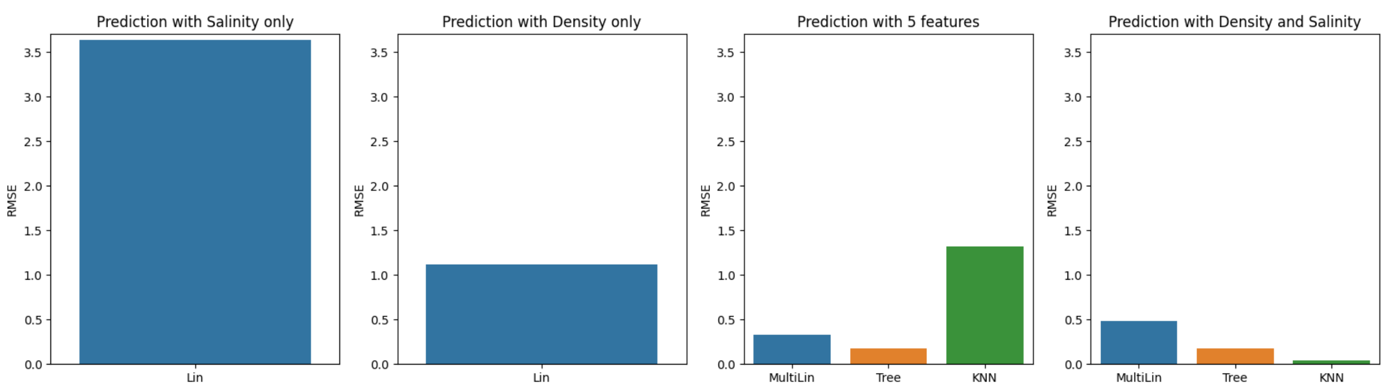
```
features2=['Salnty','STheta'] #only Salinity and Density features
X_train2 = bottle_train[features2].values
y_train2 = bottle_train[target].values
X_test2 = bottle_test[features2].values
y_test2 = bottle_test[target].values

# MULTIPLE LINEAR REGRESSION
model = LinearRegression()
model.fit(X_train2,y_train2)
y_predict2 = model.predict(X_test2)

#print(model)
#calcolo errore modello di regressione
mse_multilin=mean_squared_error(y_test2, y_predict2)
RMSE_multilin_2f=mse_multilin**(1/2.0)

print('RMSE_multilin_2f:',RMSE_multilin_2f)

#DECISION TREE
model = DecisionTreeRegressor(random_state=44,max_depth=10)
model.fit(X_train2, y_train2)
#print(model)
predictions = model.predict(X_test2)
mse_tree = mean_squared_error(y_test2, predictions)
RMSE_tree_2f= mse_tree**(1/2)
#print("MSE: ", mse_tree)
print("RMSE_tree_2f: ", RMSE_tree_2f)

#KNN
k=100
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train2,y_train2)
pred_y = model.predict(X_test2)
#print(model)
mse_knn = mean_squared_error(y_test2, pred_y)
RMSE_knn_2f = mse_knn**(1/2)
print("RMSE_KNN_2f: ", RMSE_knn_2f)

#linear regression with density
features3=['STheta'] #only Salinity and Density features
X_train3 = bottle_train[features3].values
y_train3 = bottle_train[target].values
X_test3 = bottle_test[features3].values
y_test3 = bottle_test[target].values

# LINEAR REGRESSION with density
model = LinearRegression()
model.fit(X_train3,y_train3)
y_predict3 = model.predict(X_test3)

#calcolo errore modello di regressione per densità
mse_lin_dens=mean_squared_error(y_test3, y_predict3)
```

```
RMSE_lin_dens=mse_lin_dens**(1/2.0)
print("RMSE_dens: ", RMSE_lin_dens)
```

```
RMSE_multilin_2f: 0.4813688121553125
RMSE_tree_2f:  0.1778336794244693
RMSE_KNN_2f:  0.04434452522132766
RMSE_dens:  1.1215289238624324
```

RMSE Plot

```
Metodo_g1 = ['MultiLin','Tree','KNN']
Metodo_g2 = ['MultiLin', 'Tree','KNN']
Metodo_g3 = ['Lin']
Metodo_g4 = ['Lin']
RMSE_g1 = [RMSE_multilin_2f,RMSE_tree_2f,RMSE_knn_2f ]
RMSE_g2 = [RMSE_multilin, RMSE_tree,RMSE_knn]
RMSE_g3 = [RMSE_lin]
RMSE_g4 = [RMSE_lin_dens]

#setup figure
fig, ax = plt.subplots(1,4, figsize=(20,5))
g1=sns.barplot(x=Metodo_g1, y=RMSE_g1,ax=ax[3])
g2=sns.barplot(x=Metodo_g2, y=RMSE_g2,ax=ax[2])
g3=sns.barplot(x=Metodo_g3, y=RMSE_g3,ax=ax[0])
g4=sns.barplot(x=Metodo_g4, y=RMSE_g4,ax=ax[1])
g1.set_title('Prediction with Density and Salinity')
g2.set_title('Prediction with 5 features')
g3.set_title('Prediction with Salinity only')
g4.set_title('Prediction with Density only')
g1.set_ylim(0, 3.7)
g1.set_ylabel('RMSE')
g2.set_ylim(0, 3.7)
g2.set_ylabel('RMSE')
g3.set_ylim(0, 3.7)
g3.set_ylabel('RMSE')
g4.set_ylim(0, 3.7)
g4.set_ylabel('RMSE')
#g2.set_title('Boxplot della Densità')
plt.show()
```



5 | CONCLUSION

1. Apparently there isn't a strong relationship between Temperature and Salinity as shown in the picture on the right ('Prediction with Salinity only')
2. In order to enhance the temperature prediction with the linear regression model we need to use more features (the picture in the middle): Salinity, Density, Oxygen Saturation,Depth
3. KNN model shows better results when we use the only features with a higher importance: Density and Salinity. The same features introduced from the theoretical model on Temperature-Salinity-Density

Last but not least...

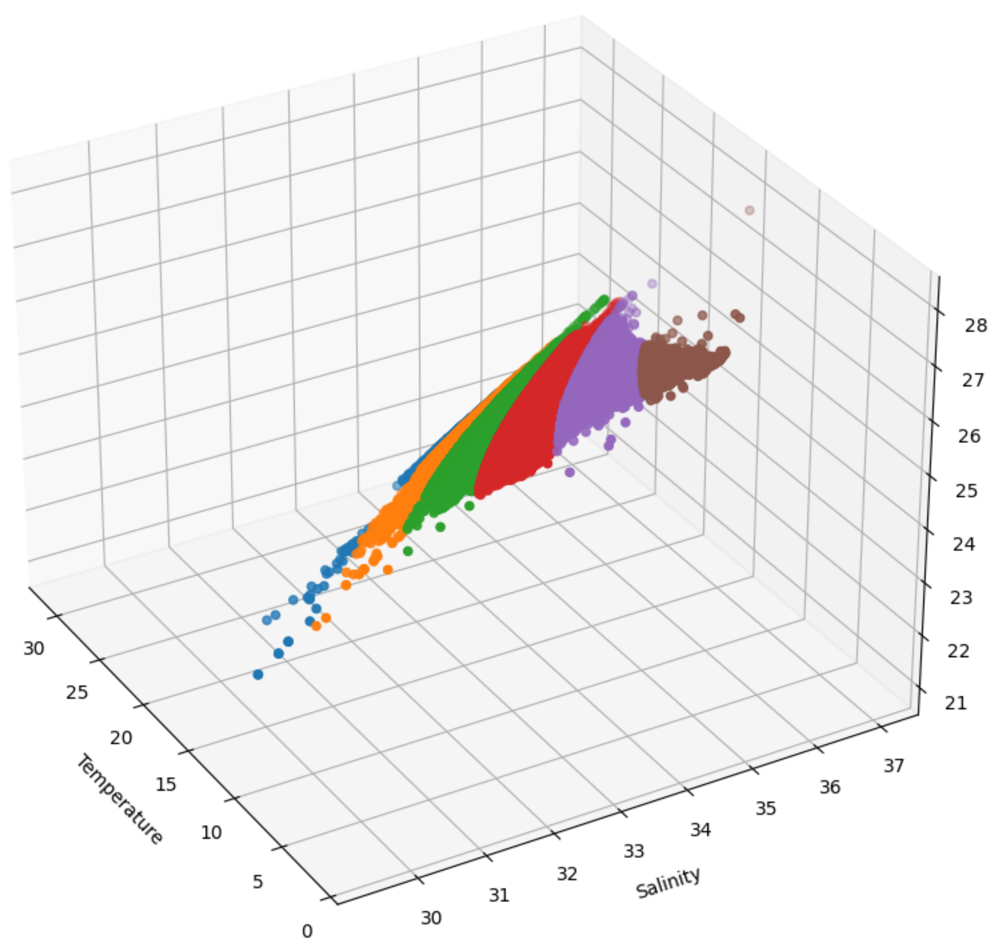
Below some plots to give a graphic explanation of why by adding the density feature we are able to improve the error.

One feature as Salinity is not enough to find a link with Temperature. A simple straight line is not able to describe a trend of a dependent variable (Temperature) function of two indipendent variables (Density and Salinity).

Instead adding one or more degrees of freedoms (i.e. Density) we give useful informations to our predictive model.

Temperature - Salinity- Density

```
theta_step=[20,23,24,25,26,27,34]
fig = plt.figure(figsize=[10,12])
ax = fig.add_subplot(projection='3d')

# set axis labels
ax.set_xlabel('\nTemperature')
ax.set_ylabel('\nSalinity')
ax.set_zlabel('\nDensity', linespacing=3.4)

for theta1, theta2 in zip(theta_step, theta_step[1:]):
    bottle_stheta=bottle[(bottle['STheta']>=theta1) & (bottle['STheta']<theta2)]
    ax.scatter(bottle_stheta['T_degC'], bottle_stheta['Salnty'], bottle_stheta['STheta'].values.reshape(-1,1))

plt.grid()
plt.gca().invert_zaxis()
plt.gca().view_init(-150, 30)
```



```
features3=['Salnty','STheta']
X_train3 = bottle_train[features3].values
y_train3 = bottle_train[target].values

model = LinearRegression()
model.fit(X_train3,y_train3)

#interpolazione dei punti del piano
N=10
x_mesh = np.linspace(30, 37, N)   # intervalli sull'asse della salinità
y_mesh = np.linspace(20, 30,N)  # intervalli sull'asse della densità
xx_pred, yy_pred = np.meshgrid(x_mesh, y_mesh)
model_viz = np.array([xx_pred.flatten(), yy_pred.flatten()]).T

predicted = model.predict(model_viz) # Valori di temperatura
```

The result of the linear regression is no longer a straight line, but a plane

```
fig = plt.figure(figsize=(10,12))
ax = fig.add_subplot(projection='3d') #crea immagine 3D

# set axis labels
ax.set_xlabel('\nTemperature')
ax.set_ylabel('\nSalinity')
ax.set_zlabel('\nDensity', linespacing=3.4)

ax.plot_trisurf(predicted, xx_pred.flatten(), yy_pred.flatten(), linewidth=0,alpha=0.6)  # plot piano di regressione
ax.set_xlim(0,30)
ax.set_ylim(30,37)
ax.set_zlim(21,28)

#Visualizzazione con view: -170,40
plt.gca().view_init(-170, 40)
plt.gca().invert_zaxis()

# plot punti del dataframe Bottle
for theta1, theta2 in zip(theta_step, theta_step[1:]):
    bottle_stheta=bottle[(bottle['STheta']>=theta1) & (bottle['STheta']<theta2)]
    ax.scatter(bottle_stheta['T_degC'], bottle_stheta['Salnty'], bottle_stheta['STheta'].values.reshape(-1,1))
```

```
fig = plt.figure(figsize=(10,12))
ax = fig.add_subplot(projection='3d')

# set axis labels
ax.set_xlabel('\nTemperature')
ax.set_ylabel('\nSalinity')
ax.set_zlabel('\nDensity', linespacing=3.4)

ax.plot_trisurf(predicted, xx_pred.flatten(), yy_pred.flatten(), linewidth=0,alpha=0.3)  # plot piano di regressione
ax.set_xlim(0,30)
ax.set_ylim(30,37)
ax.set_zlim(21,28)

#Visualizzazione con view: -170,30
plt.gca().view_init(-170, 30)
plt.gca().invert_zaxis()
# plot punti del dataframe Bottle
for theta1, theta2 in zip(theta_step, theta_step[1:]):
    bottle_stheta=bottle[(bottle['STheta']>=theta1) & (bottle['STheta']<theta2)]
    ax.scatter(bottle_stheta['T_degC'], bottle_stheta['Salnty'], bottle_stheta['STheta'].values.reshape(-1,1))
```