# 1- Oylikni Qamrab Oluvchi Yakuniy Report

1. AI bu - kompyuterning inson kabi o'ylashi, o'rganishi va qaror qabul qilishini taminlaydigan texnologiyadir. Ya'ni ma'lumotni o'rganib, muammoni yecha oladigan aqilli tizimdir.

2. ML -bu suniy intellekt sohasi bo'lib, kompyuterlarni ma'lumotlar orqali o'rganish va qaror qabul qilishga o'rgatadi. Oldindan aniq qoidalar yozmasdan. Uning ichiga **Supervise ML -**bunda input va output qiymatlar beriladi (bunda bu ham 2 ga bo'linadi: 1) **Classsification** -**discrete** (sanoqli, ya'ni chekli va shu bilan birga ham bu ikkiga bo'linishligi **binary** (ikta javob **yes** yoki **no**) va **multiclassga** (bunga **3** va **undan ortiq** javob bo'lishligi), va **Regresssion** (bu **continous** -cheksiz qiymatlar) , **Unsupervised** - bu faqat **input** qiymat berganimizda va **Reinforcement ML** asosan biz unga harakatlar orqali robot yoki robototexnikalarda bo'lishlgini bildik.

3. Project tayyorlash va uning tartibi haqida tanishdik bular:

Data Collection (Kaggle yoki Github) bunda asosan tayyor datasetlardan foydalanamiz, keyinchalik o'zimi datasetlarni tayyorlashimizni ham ko'rib chiqaiz, bu qism data bilan tanishvda ham ko'rib chiqiladi. Undan so'ng esa Data Preprocessing - bunda tushib qolgan qiymatlar ya'ni Handling Missing Valueslarni to'ldirish, bunda mean -ustundagi o'rta arifmetik qiymarlar bilan to'ldirish (faqat numerical values), mode - ustundagi eng ko'p takrorlangan elementlar bilan to'ldirish (bu har ikkisi uchun amal qiladi - numerical va categorical), median-eng o'rtasidagi element bilan to'ldirish (bunda faqat numerical), fixed - bunda o'zizmi qo'l bilan to'ldirishimiz, va asosiy shartlaridan biri soxadan habardor bo'lishimiz yoki oldin ko'rgan bo'lishligimiz muhim. Va eng ohirida 2 xil drop usulini ko'rdik bu 1-qatorlar bo'ylab drop qilish (tushirib qoldirilgan qiymatlar oz bo'lganda ishlatamiz: **df.dropna(inplace=True)**) - bu ko'd DataFrame ichidagi NaN (bo'sh) qiymatlar bor qatorlarni o'chirib tashlaydi. 2-usul esa ustun bo'yicha tashlash unda 50% va undan ko'p elementlar yo'q bo'lsa uni to'ldirish o'ta suniylashib ketkani sabab yaxshisi o'sha ustundan qutulish foydaliroq deya foydalamiz:
**df.drop('Exercise Angina', axis=1, inplace=True)** deya.

O'rtada biz **kutubxonalar** bilan tansihdik bularga: **Numpy** (statistika va matematika boshqaruvi va taxlil qlish), **Pandas** (**Ma'lumotlarni kirgazish**, **Ma'lumotlar bilan tanishish**, **Ma'lumotlarni o'zgartirish** va **tozalash** va **Ma'lumotlarni taxlil qilish**), **Matplotlib** (**Grafiklar** va **Dizaynlar)**, **Seaborn** (**Matplotlibni** ustida qurilgan chizmalarni ozgina chiroyli qilib (**vizual ko'rinishda**) beradigan kutubxona , **Scikit-learn** - bu **ML o'qitish**, **ML project bosqichlarini bajarish** va bu hatto **Supervised MLning yuragi** desak ham bo'ladi.

**Dataga** tavsif ham berdik bunda: u **to'liq**, **katta**, **soxaga aloqador** va **ishonchli bo'lishligi** zarurligi.

**Encoding** haqida ham muhim ko'nikmalarni o'rgandik va ta'rif berdik: Birorta so'z ko'rinishdagi berilgan datasetlarimiz ya'ni umumiy **categorical** yoki **stringni numerical** holatiga o'tkazish jaroniga aytilishini y'ani --> **Encoding** --> **Categorical** ---> **Numerical (int64, float64).**

Va turlari; **one-hot** - bu kategoriayli (matnli) ma'lumotlarni mashina o'rganish modellari tushunadigan ikkilik (0 va 1) ko'rinishiga o'tkazish usuli va unda har bir kategoriya alohida ustun bo'ladi (mos kategorya 1 qolganalri 0): **dummies = pd.get_dummies(df['size'], prefix='col', dtype=int) va df = pd.concat([df.drop(columns=['size']), dummies], axi=1).** **Label** (categorcail qiymatlarni alifbo tartibida ketma-ketlikda 0dan boshlab z gacha neechta qiymat bo'lsa o'shatgacha almashtiradi): f**rom sklearn.preprocessing import LabelEncoder, encoder = LabelEncoder(), encoder** --deya chaqirib olamiz va so'ngra **df['player_name'] = encoder.fit_transform(df['player_name'])** deya ishlatamiz va threshold haqida ham so'z ketdi bu bizga chegarani qo'yishimizga yordam beradi.

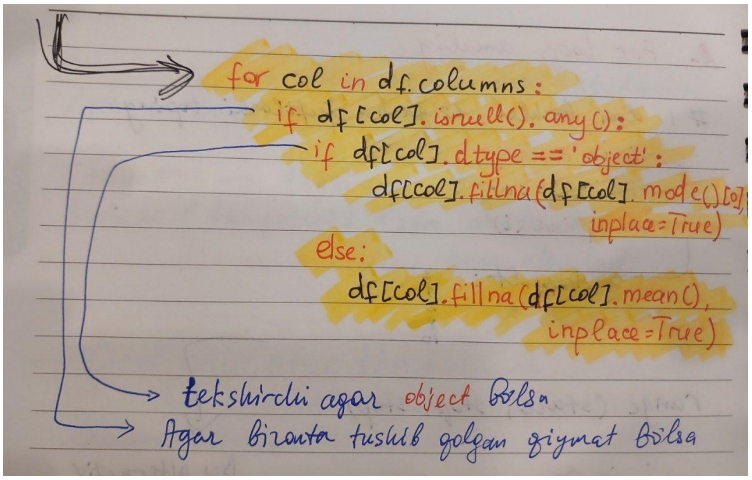
**frequency** (takrorlanishalr sonini umumiy elementlar soniga nisbati bilan almashtiriladi. Masalan mushuk 2 marta takorolangan va umumiy elementlar soni 4 ta shunda 2/4 ga bo'lsak 0.5 javobi chiqadi) , target (Mos target qiymatning o'rta arifmetigiga ko'ra encoding qiladi (Masalan mushukning mos targetdagi qiymati 0 va 1 uni qo'shadi 0+1=1 endi uni nechi marta takrorlandi 2 marta shunga bo'ladi javob 1/2 bo'lsa 0.5) va ordinal (bu tartibga ega categorical elementlarni sonlar bilan almashtirishdi masalan o'chamni

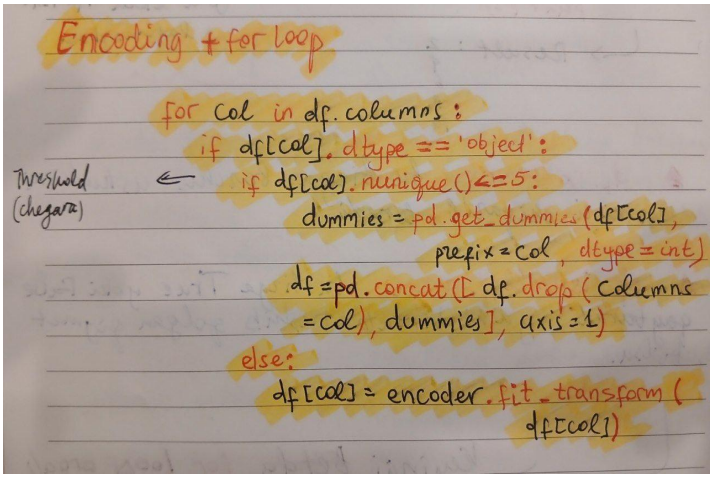
olsak unda **xs-1, s-2, m-3, l-4, xl-5** … shunday ko'rinishda) encodinglardan iborat ekanligi. Shuning orasida For Loopgaga ham teginib o'tdik bunda: **For Loop** bu jarayonlarni avtomatlashtirish uchun ishlatilinadigan kodlar bloki bunda **for - keyword**, i **- esa o'zgaruvchi** : **for i in range(2,11):**

**if I % 2 == 0:**

**print(I) --> Result: 2,4,6,8,10**

Undan so'ng esa **df.columns** - bu bizga barcha ustunlarni nomini chiqarib berishi, **df.isnull().any()** esa bizga True yoki False qaytarishi ya'ni bironta tushib qolgan qiymatlar mavjud bo'lganda. Va shular orqali biz for loopda jarayonni avtomatlashtirshni urg'u berdik.




Ma'lum bir bosqichlarni tashlab ketsak ham ma'lum bir algoritmlarda kodlarimiz bexato ishlashi mumkin, ammo ikta xolatda to'xtashi mumkin bu 1) Ko'd o'zi to'xtaydik va ekranga yozuv chiqadi- **Error** degan ya'ni senda tushib qoldirilgan qiymat bor deya aniqrog'I **NaN** bor deydi va to'g'irla deydi.
Holatda esa ba'zi algoritmlar-yana ham aniqroq aytganda kuchliroq algoritmlar tushurib qoldirilgan qiymatlarni ham farqi yo'q ishlayveradi. Ammo, tushirib qoldirilgan qiymatlardagi joyi kemtik bo'lib qolaveradi. Ya'ni o'sha joyidan muammo yuza chiqib kelaveradi. Bu esa pirovardida **prediction** qilish qismida yoinki natijinani olish qismida bizga kutilgandek natija qaytarmaydi. Shuning Data Preprocessing qismi juda muhim bosqichlardandir.

**Scaling -** Juda katta va juda kichik qiymatlar ortasidagi balansga olib kelishdagi Data Preprocessing bo'limidir. Bunda ustunlar numerical bo'lishi va qiymatlarimizni agar [0,1] oraliqda bo'lsa scaling qilish shart emsaligiga alohida e'tibor beramiz. Scaling qilishdan oldin va keyin ham df.head() qilishimiz maqsadga muvofiq sababi xato qilmabmizmi yoki yo'qmi, ko'dimiz ishladiimi yoki yo'qmi degan muhim savollarga javob olamiz.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
```

Shu orqali 3 xil scaling classlarini import qilib olamiz.

```
num_col = df.select_dtypes(include=['int64', 'float64']).columns.drop('Personal Loan')
# Izoh: --> Target qiymat scaling qilinmaydi.
```

Bu yerdagi asosiy e'tibor berildigan jabhalardan biri bu numerical qiymatlarni o'zingizning datasetingizdan kelib chiqib **dtypelarni** yozasiz **include** ichiga. Masalan, int64, float64 va hokazo. Va har birini bittadan yozsangiz kifoya. Shuningdek, Target qiymar scaling qilinmaydi.

```
scaler = StandardScaler()
scaler
df['Age'] = scaler.fit_transform(df[['Age']])

df.head() # Bu yerda StandardScaler bo'lgani sabab bizda manfiy tarafga ham o'tkazib yubordi.
```

Bu yerda iktalik qovus ichiga agar faqat bir ustunni sclaing qilmoqchi bo'lsak yozamiz. Agar 2 va undan ortqi ustunlarni scaling qilmoqchi bo'lsak 1 ta to'trburchak qovus kifoya qiladi.

```
scaler = MinMaxScaler()
scaler
df['ZIP Code'] = scaler.fit_transform(df[['ZIP Code']]) # Python case sensitive va asosan musbat ko'rinishda [0,1] orqaliqqa olib keladi.
```

```
# RobustScaler

scaler = RobustScaler()
scaler
df['Family'] = scaler.fit_transform(df[['Family']])
df.head() # Bu yerda Familyni ham manfiy ham musbat tomonga o'tkazib yuboradi.
```

For Loop + Scaling;
Bunda biz jarayonni **avtomatlashtirish** uchun ishlatamiz va **umumiy** yoki **guruhlarga** bo'lib **scaling** qilishmiz uchun ishlatilinadi.

```
scaler = MinMaxScaler()
for col in df.columns:
    if df[col].dtype != 'object':
        df[col] = scaler.fit_transform(df[[col]])
```

**Yoki**

```
df[num_col] = scaler.fit_transform(df[num_col])
#numerical columns orqali scaling qilish
num_col = df.select_dtypes(include=['int64', 'float64']).columns.drop('Personal Loan')
# Izoh: --> Target qiymat scaling qilinmaydi.
for col in num_col:
    df[col] = scaler.fit_transform(df[[col]]) # bu ham bir variant scaling qilish uchun
```

Va so'ngida biz data structure haqida gaplashdik bunda data(Raw Data) - ya'ni fan tilida hom ashyo undan so'ng Data bilan tanishuv bunda bazi predictive kuchga ega bo'lmagan

namelarni tashlab yuborsak ham bo'ladi. Data Preprocessing (Handling missing values, Encoding va Scaling), undan so'ng Model selection (algorithm -> method) va uning datadan kelib chiqib qaysi biri mos kelishi masalan Linear uchun Linear Algorithm, Non-Linear uchun esa Complex Algorithms ishlatilishi kerak bo'lishligi haqida ma'lumot berildi.

Unda so'ngra biz **ML Model Structure** haqida gaplashdik unda: **Data Collection, Data bilan tanishuv va Data Preprocessing**larni tugatganimiz va so'ngra quyida qolgan qismlari bular; **Model (Algorithm) Selection, Model Training va Evaluation**larga yana chuqurroq urg'u berdik.

Model va algoritmga tushunarliroq tarif berish o'laroq uni yaxshiroq hayotiy o'rinda tushunishga harakat qildik masalan Modelni bir ovqat desak, algoritm bu retsept ya'ni ovqatni qancha vaqtda pishirish, qancha va nima qo'shish, qanday ketma-ketlikda bajarish, qancha qovurish va hokazolar. Yoki o'qituvchi bilimi bu model lekin uni o'quvchiga moslashi ya'ni o'yin tarzida, video ko'rish orqali yoki og'zaki imtixon qilish orqali yoki jamolarga bo'lib o'rgatishni misol qilib aytsak bo'ladi algoritm uchun.

**ML algorithm family (Supervised ML)** mavjud bo'lib aslida buni biz o'zimiz yaxshiroq eslab qolish uchun oila tarziga keltiib oldik. Bunda **Supervised ML** algoritmlar juda ko'p bo'lib, ularni kodlari bilan eslab qolish mushkul ishdir. Shuning uchun biz ularni kutubxonalaridan chaqirishim ancha qulaydi.
Bularga; **Linear (Linear Regression, Logistic Regression), Tree-based (Decision Tree, Random Forest), Distance-based(KNN, SVM) va Ensemble(Random Forest, Gradient Boosting)** lar kiradi. Data Preprocessing orqali tayyorlab olgan datasetimizni kompyuterga tuhsunarli qilib o'rgatish albatta har hil natijalar beradi. Loyihamizda kelgusida '**Comperative Evaluation'** qismi bo'ladi unda bir nechta algoritmlar bilan ishlatib ko'rib unga baho beramiz. Va o'sha algoritm bilan model yasaymiz, proyekt quramiz.

Quyida kodlardan parchlar keltirilgan;

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder
```

```
▾ LabelEncoder ⓘ ⓘ
▸ Parameters
```

```
from sklearn.linear_model import LinearRegression,
LogisticRegression # bu yerda LinearRegression bu faqat
Regression uchn ishlaydi, LogisticRegression esa faqat
Classification uchun ishlaydi.

lr = LinearRegression()
lr
```

```
▾ LinearRegression ⓘ ⓘ
▸ Parameters
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestRegressor
```

Liner Regression bu baseline modellar hisoblanadi. Ya'ni bir boshidan baseline model qurib olib, kuchsizroq model qurib keyin uni rivojlantirishga harakat qilamiz, Ya'ni o'zimi bir planka qo'yamiz va plankadan oshishga harakat qilamiz.

Bunda Classification-> Logistic Regression (mantiq funksiyasi, asosida ishlaydi masalan (0,1) ya'ni discrete - chekli 2 ta qiymatdan iborat), Regression-> Linear Regressiondir.

Logistic Regression - bu classification uchun ishlatilinadi va asosan binary classification'larda yaxshi ishlaydigan algoritm hisoblanadi. Bunda e'tibor berishimiz kerak bo'lgan 3 jihat bor; Logistic Regression, classification va Regression uchun ishlamasligi.

Classification tasklar uchun effektiv algoritmlardan biri; Bunda sodda (ya'ni unchalik katta yoki kompleks bo'magan loyihalarda ishlatsak ham bo'ladi), kichik datasetlarda yaxshi ishlaydi, va Binary classificationda yashi ishlaydi (2 ta klasslarda iborat bo'lganda)

Logistic Regressiondagi bosqichlar quyidagilardan iborat; Data Preprocessing, (x,y), Splitting (train,test), Training(fitting), Predicting va Evaluating.

Model trainingda umumiy scaling jarayonida quyida formula orqali bajariladi; $\mathbf{x_{scaled}} = \frac{x - xmin}{xmax - xmin}$ . Bazida bu holatda garchi ustunlar int bo'lsa ham scaling qilishda floatga o'tib qolishi mumkin.

Keyingi bosqichlarda esa biz ma'lumotlarni ikkiga ajratiib olishimiz kerak. Bamisoli o'qituvchi o'quvchilarga imtixonga tayyrolash uchun asosiy bilimlarni ko'p beradi va imtixonda esa bergan bilimiga qaranga ancha kam hajda uni imtixon qilib oladi. Bunda Train uuchun datasetimizdan kelib chiqib 70% yoki 80% olamiz, Test uchun esa 30% yoki 20% olamiz. Va bu aynan Splitting deyiladi. Aslida Splitting qilish bir muncha qiyin va vaqt olganligi sababli tayyor kutubxonadan fodalanish bizga ancha qulaylik keltiradi.

```
from sklearn.model_selection import train_test_split # buning asosiy vazifasi 2 ga ajratib berish
# inputlarni olib qolishni oson yo'li bu outputni drop qilish yetadi.

x = df.drop(columns=['annual_cost_healthy_diet_usd'])
x = df.drop(columns=['country_code']) # bu esa datasetimizda precit uchun muhim bo'lmagan ustundan qutulish mumkin
```

```
y = df['annual_cost_healthy_diet_usd'].astype(int) # buni qilishdan maqsad keyinchalik x_test qilganda bizga chiqaradigan qiymatimiz float emas int chiqqani maqulroqligi sabab

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # shu xolatda random_state=42 modelni balansli ishlashi uchun.
```

Bu yerda test_size =0.2 degani 20% berildi qolgani avtmotik tarzda trainga beraman degani.

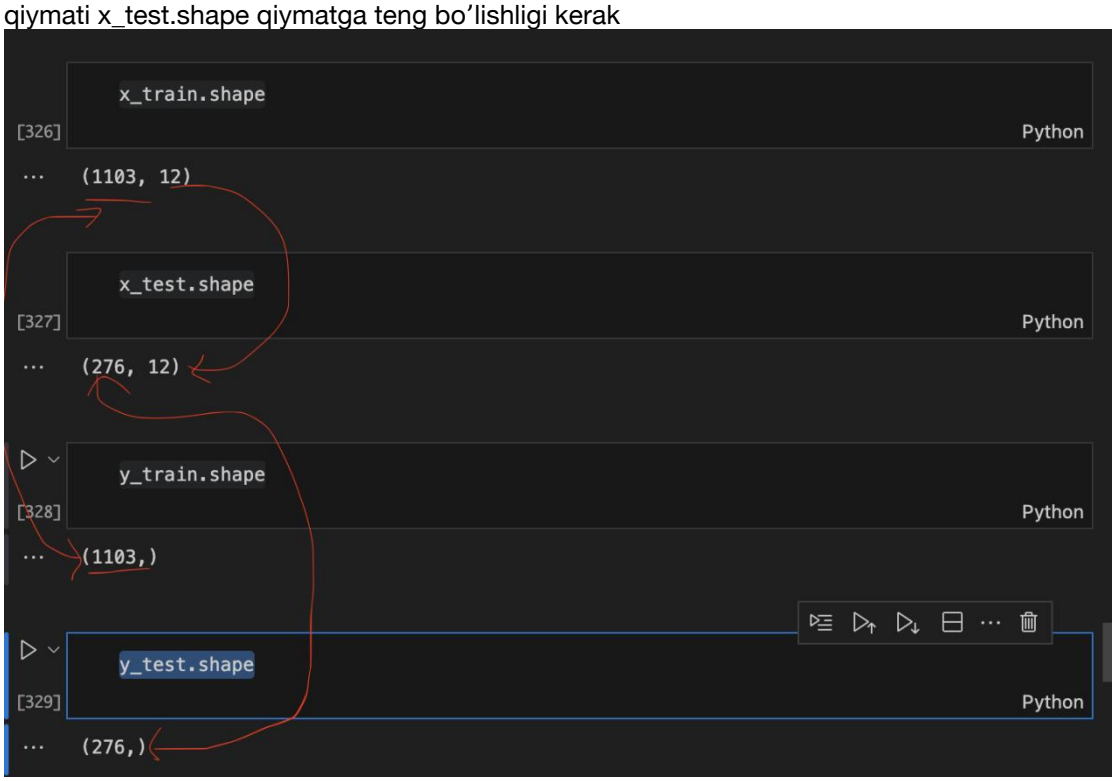
```
x_train.shape
```

```
x_test.shape
```

```
y_train.shape
```

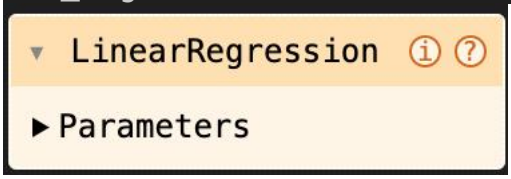
```
y_test.shape
```

Bunda es ax_train.shape qiymati y_train.shape bilan bir xil va x_train.shapening ikkinchi qiymati x_test.shape qiymatga teng bo’lishligi kerak




```
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()
lin_reg
```




Bunda not fitted bo’lgan edi ya’ni hali ishga tusmaganligini bildiradi.

```
lin_reg.fit(x_train,y_train) # bu yerda traininglar bilan train ishlatilinadi.
```

```
▾ LinearRegression ⓘ ?
▸ Parameters
```

Bunda esa rangi ko'k bo'ldi bu esa fitted yani ishga tushirilganini bildiradi.

```
y_pred = lin_reg.predict(x_test) # aqilli qilib olgan testimizni predict qilamiz
y_pred[1:10]
```

```
array([-0.00064414, -0.0171581 ,  0.00195075, -0.00100796,  0.00553704,
       -0.00094837,  0.00534536, -0.00096763, -0.00038961])
```

Endi modelimiz qanchalik yaxshi ishlashini qayerdan bilamiz? Bunianiqlash uchun bizga metrics kerak bo'ladi. Ya'ni metrics orali baholaymiz modelimiz yaxshi ishlayaptimi yoki yo'q.

Mening datasetimiz LinearRegreesionda ishlagani sabab men mean_absolute_error orqali tekshirdim. Agar Logistic Regression bo'lganida biz accruacy_score olgan bo'lar edik.

```
from sklearn.metrics import accuracy_score
```

Menda 0.00579.. atrofida chiqdi bu holatda **mean_absolute_error** qanchalik past chiqsa shunchalik yaxshi ekan.

```
mean_abs_error
```

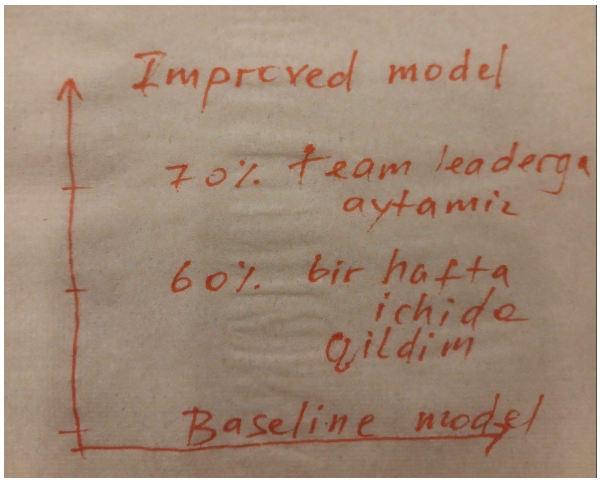
```
0.0057918175403781896
```

So'ngida esa biz **Logistic Regression** ko'proq **binary classification** uchunligi haqida gaplashdik va **multiclasslar** uchun esa **Decision Tree** ishlatish yaxshiroqligi haqida ham gaplashdik.

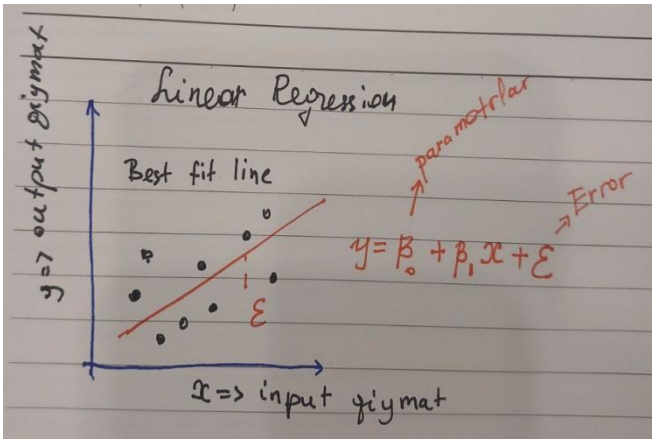
**Training** jaroyoniga ham to'hdaldik bunda; **Preprocessing** qilingan datani biror algoritm yordamida kompyuterning miyyasiga quyish yoki o'rgatish hisoblanadi.

Kompyuter datalar yordamida o'rganadi ma'lumotlar oladi, **patternlarni** o'rganadi shunga qarab turib qaror qabul qiladi.

**Linear Regressionga** quyidagilar kiradi:
- Linear family,
- Baseline model
- Regression (continues)
  * Players goals
  * Net injury
- Linear eqaution **( y = kx + l)**



**Linear Regression**ning ko'rinishi bunday:




Masalan bunga misol o'laroq **osh** tayyorlashni olib unda; osh ---> u output, qolganlari ya'ni masalliqlar o'lchovi esa bu inputlardir.

**Parametrlar** **input** qiymatni muhimligini o'lchovchi o'lcham hisoblanadi.

Linear Regressionda demak; **x-->** **input**, **y** --> **output**, **betta** --> **parametr**, va **epsilon** --> **Error**

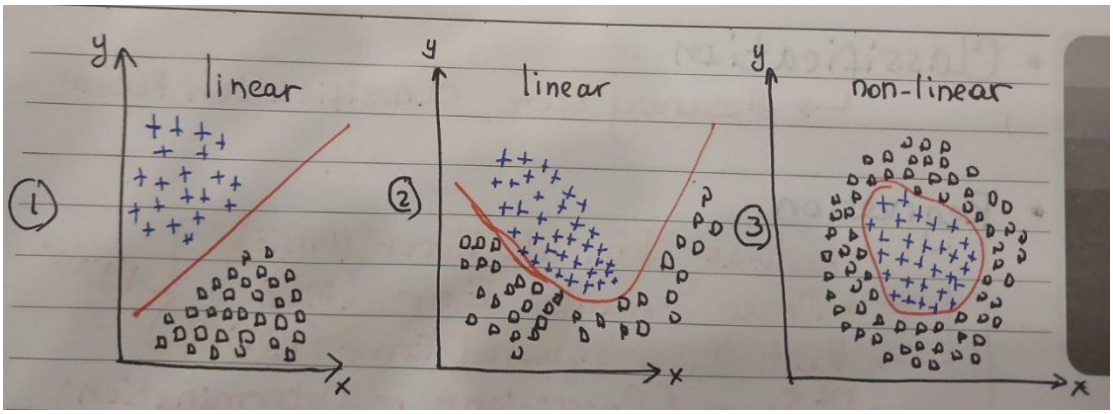
**Linear Regression** maqsadli yoki Linear Regression ishlatayotkanimizda natija zo'r chiqayotkan bo'lsa qachonki liniya chiqiz paydo bo'lsa (input va output uchun mos chiqiz topib berish). Linear Regression o'z navbatida: **Simple Linear Regression** va **Multiple Linear Regression** ga bo'linadi. Bunda featurlar ko'p, juda ko'plab input qiymat uchun parametr qo'shiladi.

Qo'chimcha qiladigan bo'lsak --> **Epsilon** bu **Error** hech qachon 0ga teng bo'lmaydi (**!=0)**. Sababi, **Error** kamaytirishga harakat qilinadi lekin albatta **epsilonimiz 0** ga teng bo'lmaydi. Buni ham hisobga olish kerak.

Va **Linear Regressionni pros** va **conslari** mavjuddir.

- **Pros**:
  - *** Tez va soddadir**
  - *** Kichikroq datalar bilan ishlashi oson**
  - *** Baseline model qurish uchun qulayligida**
  - *** Kam chiqim (low cost) - oz material va harajat talab qiladi.**

- **Cons**
  - *** Complex (non-linear) xolatlarni o'rgana olmaydi.**
  - *** Yod elementlarda tez o'zgaruvchan (stabil emas)**
  - *** Katta datasetlarda yaxshi ishlamaydi.**




Yuqoridagi rasmda esa ko'rishmiz mumkin Linear Regression va Non-Linear Regression uchun misolni.

**Evaluation** (baholash) --> **Generilization** va **Overfitting**

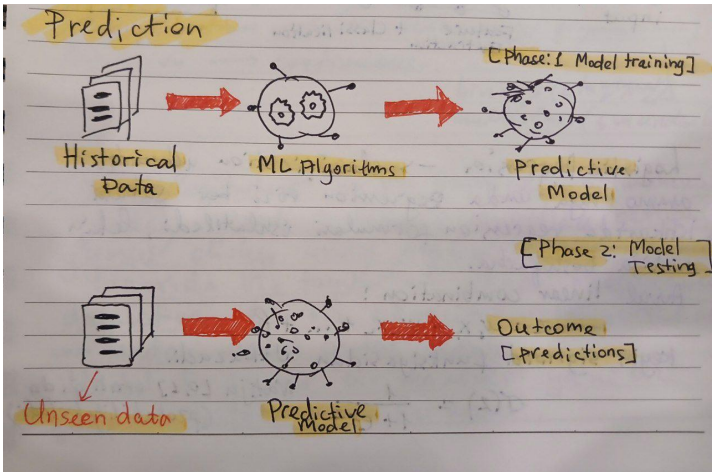
Quyidagilarni baholaymiz: - Prediction qancahlik yaxshi? (raqam bilan o'chaydi masalan [**0, 100]** oraliqda), - **Model** ishinarlimi? (Biz qancahlik ishonishimiz mumkin? ), - **Generilization** (Yaxshi umumlashtirdimi?)

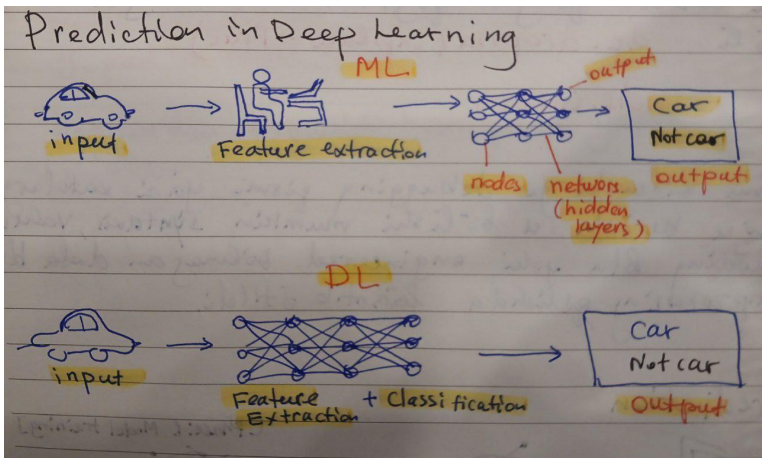
**Overfitting** (bu yomon so'z ekan)- bu model 'train data'ni haddan tashqari yodlab olishdir. Masalan, talaba savolni tushunmay javoblarni yodlab olishi, ammo unga raqamlarni almashtirib berilsa javob topa olmasligi. **Generilization** (bu yaxshi so'z ekan) esa umumlashtirgan, mantiqan yondoshgani uchun buning javobi bu chiqadi deya aniq biladi talaba.
So'ngida esa **Classification** uchun biz --> - **Accuracy score** va **Classification Reportdan.**
**- Regression** uchun esa biz --> **Mean Absolute Error** (**MAE**), **Mean Squared Error** (**MSE**), **Root Mean Squared Error** (**RMSE**) va **$R^2$ Score** (**Coefficient of Determination**) lardan foydalanishimiz mumkin.

Bunda asosiy e'tibor beradigan jihati shundagi birichi 3 tasi qancha kichik chiqsa shuncha yaxshi, sababi u origianl **$y_{test}$ [0] = 1** va **$y_{pred}$[0] = 0,1** o'rtasidagi farqni topadi.

**$R^2$ Score** esa teskarisi, qancha yuqori chiqsa shuncha yaxshi.

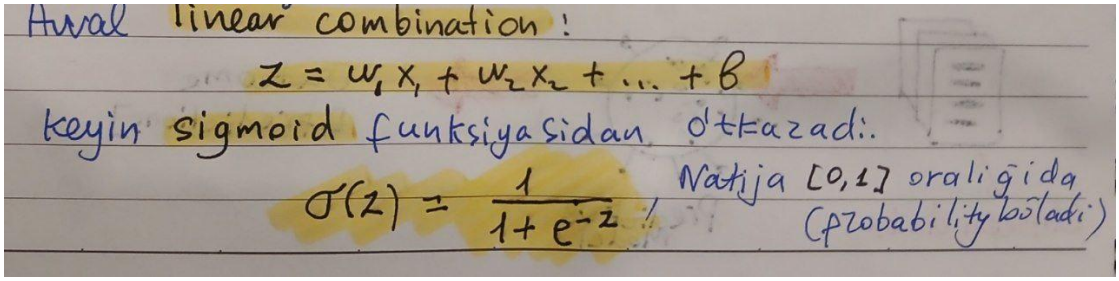
Shu bilan birga bizga darsning so'ngida masalahatlar berildi bulardan: Ioji bo'lsa Regression uchun Classificationni ishlatib ko'rish undagi xatoliklarni ko'rish va undan o'rganish kerakligi, Improvement oyligida esa Errorlarni kamaytirish va r2_scoreni oshirish haqida, Predcition bo'limini mag'zini tushunishga e'tibor berdik bunda trained qilingan modelni unsean data bilan predict qilish ( **lr_model = LinearRegression(), lr_model.fit(x_train, y_train), y_pred = lr_model.predict(x_test)** ) larni ko'rdik. Hatolardan o'rganish ham juda yaxshi o'rganish usulidir. Masalan **ValueError**: c**ould not convert string to float: 'Cardiology'**--bundan esa Agar biz **encoding** qilmasak bizda **trainda** shu kabi **Error** berishligi mumkinligi, Ammo bazida **model trained** bo'lib garchi qiymatlar tushsa ham ishlab ketishligi mumkinligi, ikki xolatda vaziyatga nigoh soldik 1) **Coding Error** berishligi, 2) **Model** pirovardida yaxshi natija bermasligi. Bazi algoritmlar bor bizda **scaling Tree-based** algoritmlar bo'lsa uni **scaling** qilish shart emas sababi T**ree-based structure** boshqacha. **Linear algoritmlar** bilan ishlayotkanimizda **Logistic Regression**, **Linear Regression** bilan ishlayotkanimizda biz albatta **scaling** qilishimiz zarurligi. Agar **scaling** qilmasak ohirida **y_pred = lr_model.predict(x_test)** da pand berib natijani biz kutgandek bermasligi. Va agar **datasetimizda** oz tushib qolgan qiymatlarni tashlashni eng yaxshi usuli deya bu **df.dropna(inplace=True)** dan foydalanganimiz. Va **Predcitionga ML** va **DL** nigohida qaradik:




Bu yerda biz asosiy g'oya **train (x_train, y_train)** datasetlar orqali train qilamiz va **predictionni x_test** orqali qilamiz va **x_testimiz** bizda **unseen** data ya'ni ko'rilmagan data hisblanadi.

Shu yerda bir qiziq holat ko'rinadi, masalan **Logistic Regression** -> **classification** uchundir ammo nega unda **Regression** so'zi bor sababi ichkarida regression formulasi ishlatiladi, lekin natija boshacha:




Yakunlovchi fikr o'laroq bizga **"No Free Lunch Theorem"** - **Odatda bitta eng zo'r algoritm yo'q ya'ni hamma muammo uchun bir xil yaxshi ishlaydigan model bo'lmaydi** degan ML ko'p ishlatiladigan katta konsepsiya mavjudligi takidlandi.