1회차 스터디 서기

*데이터 분석 - 쉽게 시작하는 캐글 데이터 분석

💡 팀 이름: 캐굴캐굴
회차: 1회차
서기 일자: 3/16
서기 : 김하원
시간: 3/16 11:00 ~ 13:00
장소: 노량진역

▼ 출석 사진

[시작]

12:59(3인)

11:08(전원 도착)

[종료] 12:57

▶ 스터디 목차

1. 타이타닉 생존자 예측 경진대회란?
2. 데이터 개요 파악
3. 데이터 시각화
4. 데이터 전처리 및 특징 값 생성

1. 타이타닉 생존자 예측 경진대회란?

- 승객의 성별과 연령, 승선 티켓 등급, 생사 여부 등 여러 정보가 포함
- 승객의 정보를 통해 생사에 영향을 주는 속성이 무엇인지 파악 가능
  - **승객 정보(연령, 티켓 등급 등) → 생사 여부**
- 해당 데이터는 데이터 분석의 벤치마크 데이터셋으로 유명

  ▼ 벤치마크 데이터셋이란?

  Re: 모델의 품질을 평가하기 위해 설계된 평가 방식
  - 따라서 해법과 정답이 모두 공개되어 있음(연습용 데이터)

---

2. 데이터 개요 파악

- **탐색적 데이터 분석(EDA):** 그래픽과 시각화를 이용하여 데이터 집합을 탐색하고 분석하는 일. 목표는 통계적 가설을 확인하는 것이 아니라 탐색, 조사 및 통찰. +) 결측치, 이상치 파악에 용이

▼ 질적변수와 양적변수

- 질적변수 (범주형 변수): 수량화 할 수 없고 범주(category)로 구분하는 변수
  ex) 성별, 직업, 종교 등
  - 명목 척도: 범주간 크기의 순서가 존재하지 않음 ex) 종교, 성별
  - 순서 척도: 범주에 순서가 존재함 ex) 상중하 평가
- 양적변수 (연속형 변수): 수치로 나타낼 수 있는 변수
  - 간격 척도: 각 수준 간 간격이 동일한 척도, 절대적 0점 존재하지 않음, 단위가 부여되지만
    비율은 의미가 없음, 곱하기 불가능 ex) 온도, 좌표
  - 비율 척도: 절대적 크기 비교, 비율에 의미가 있음, 곱하기 가능 ex) 질량

▼ 관련 코드

```
print(train_df.shape) #행과 열 확인
print(test_df.shape)

train_df.head()
test_df.head()

train_df.dtypes #데이터 속성확인

train_df.describe() #데이터 통계량 확인
test_df.describe()

train_df["Sex"].value_counts() #카테고리 변수 확인
train_df["Embarked"].value_counts()
train_df["Cabin"].value_counts()

train_df.isnull().sum() #결측치 확인
test_df.isnull().sum()
```

3. 데이터 시각화

- **Matplotlib 표시 스타일 종류**

  1) default

  기본 스타일로, Matplotlib의 기본 스타일을 사용

  2) classic

  전통적인 스타일로, 이전 버전의 Matplotlib과 비슷한 스타일 제공

  3) ggplot

  R의 ggplot 스타일과 유사함, default와 유사하나, 객체의 선택이 자유롭고 다채로움

  4) seaborn

  Seaborn 라이브러리의 스타일을 따르며, 색상 팔레트와 스타일을 설정하여 매력적인 그래프 생성이 가능

  5) bmh

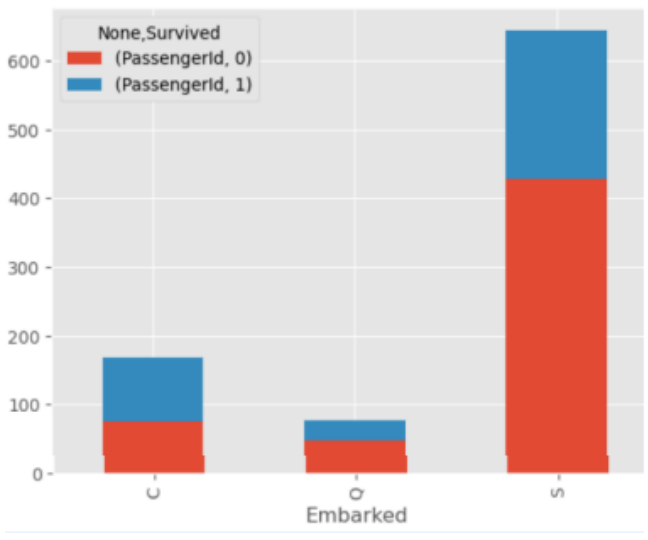
  Bayesian Methods for Hackers 스타일로, 확률론적 그래프의 디자인을 제공

- **목적 변수 관련 데이터 전처리**
  - isnull: 행과 열 별로 결측치의 존재 여부 판정 가능
  - sum(): 각 행의 결측치 값의 합계
  - dropna(): 결측치 제외
  - groupby(['집계하고 싶은 열 명']).집계 함수: 특정한 카테고리로 그룹화
  - 데이터 수평 변환
    - unstack(), 수직 정렬: 행을 열로 변환, 행 인덱스의 최하위 레벨을 열 인덱스의 새로운 최하위 레벨로 이동 → 행 인덱스 감소, 열 인덱스 증가
    - stack(), 수평 정렬: 열을 행으로 변환, 열 인덱스의 최하위 레벨을 행 인덱스의 최하위로 이동 → 열 인덱스가 감소, 행 인덱스가 증가

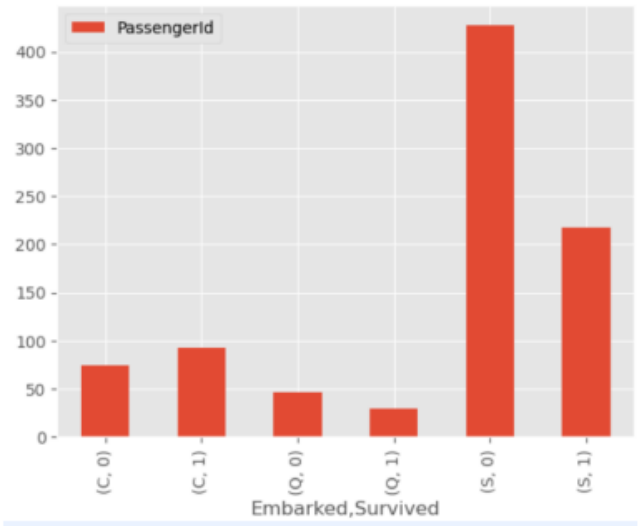
▼ 관련 코드

```
train_df.isnull().sum() #결측치 개수 확인
test_df.isnull().sum()

train_df[["Embarked","Survived","PassengerId"]].dropna() #결측치 제거
train_df[["Embarked","Survived","PassengerId"]].dropna()
.groupby(["Embarked","Survived"]).count() #group화 하여 집계

embarked_df = train_df[["Embarked","Survived","PassengerId"]].dropna()
.groupby(["Embarked","Survived"]).count().unstack() #df를 수평으로 변환
```

- 데이터 프레임 수평 수직 정렬 시 X축 변화
  - 수평 정렬 시 - x축 Embarked



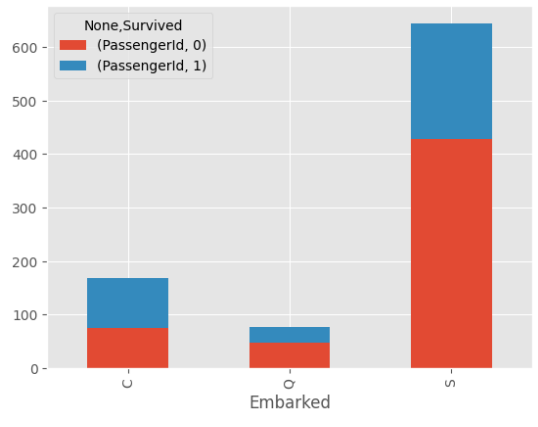
  - 수직 정렬 시 - Embarked, Survived



- **탑승 항구별 생존율 시각화**
  - df.iloc[행 번호, 열 번호]: 특정 행, 열 추출
  - df.loc[행 명, 열 명]: 특정 행, 열 추출

▼ 1) 탑승 항구별 생존율 시각화

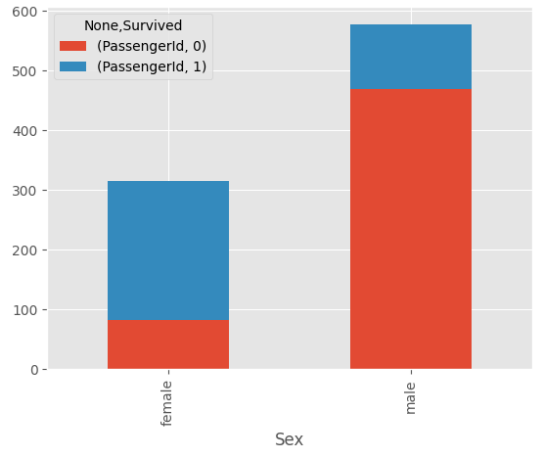
```
embarked_df = train_df[['Embarked', 'Survived', 'PassengerId']].dropna()
.groupby(['Embarked', 'Survived']).count().unstack()
embarked_df.plot.bar(stacked = True)
```



C 항구에서 사망률이 가장 낮음

▼ 2) 성별별 생존율 시각화

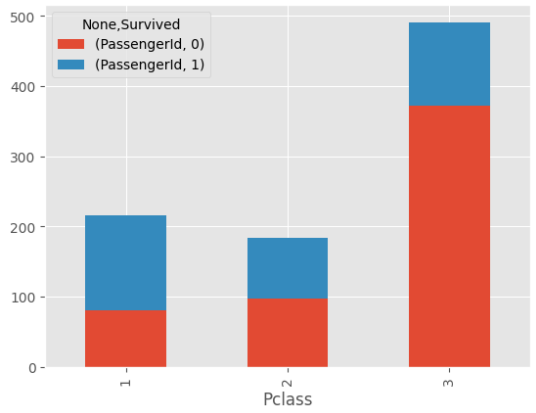
```
sex_df = train_df[['Sex', 'Survived', 'PassengerId']].dropna()
.groupby(['Sex','Survived']).count().unstack()
sex_df.plot.bar(stacked=True)
```



여성의 생존율이 남성에 비해 높음

▼ 3) 티켓 종류별 생존율 시각화

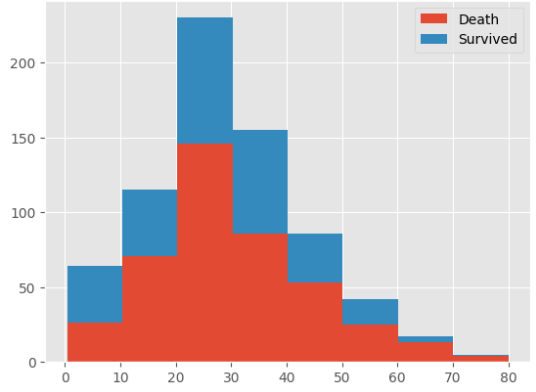
```
ticket_df = train_df[['Pclass','Survived','PassengerId']].dropna()
.groupby(['Pclass','Survived']).count().unstack()
ticket_df.plot.bar(stacked=True)
```



3등급, 2등급, 1등습 순으로 생존율이 높음

▼ 4) 연령별 생존율 시각화

```
plt.hist(x=[train_df.Age[train_df.Survived ==0], train_df.Age[train_df.Survived == 1]],
          bins = 8, histtype='barstacked', label = ['Death','Survived'])
plt.legend()
```



60대 이후 연령층의 생존율이 타 연령층에 비해 낮음

- **상관계수를 활용한 히트맵 제작**
  - 카테고리 변수를 더미 변수화하는 과정이 필요 → **원-핫 인코딩을 사용**

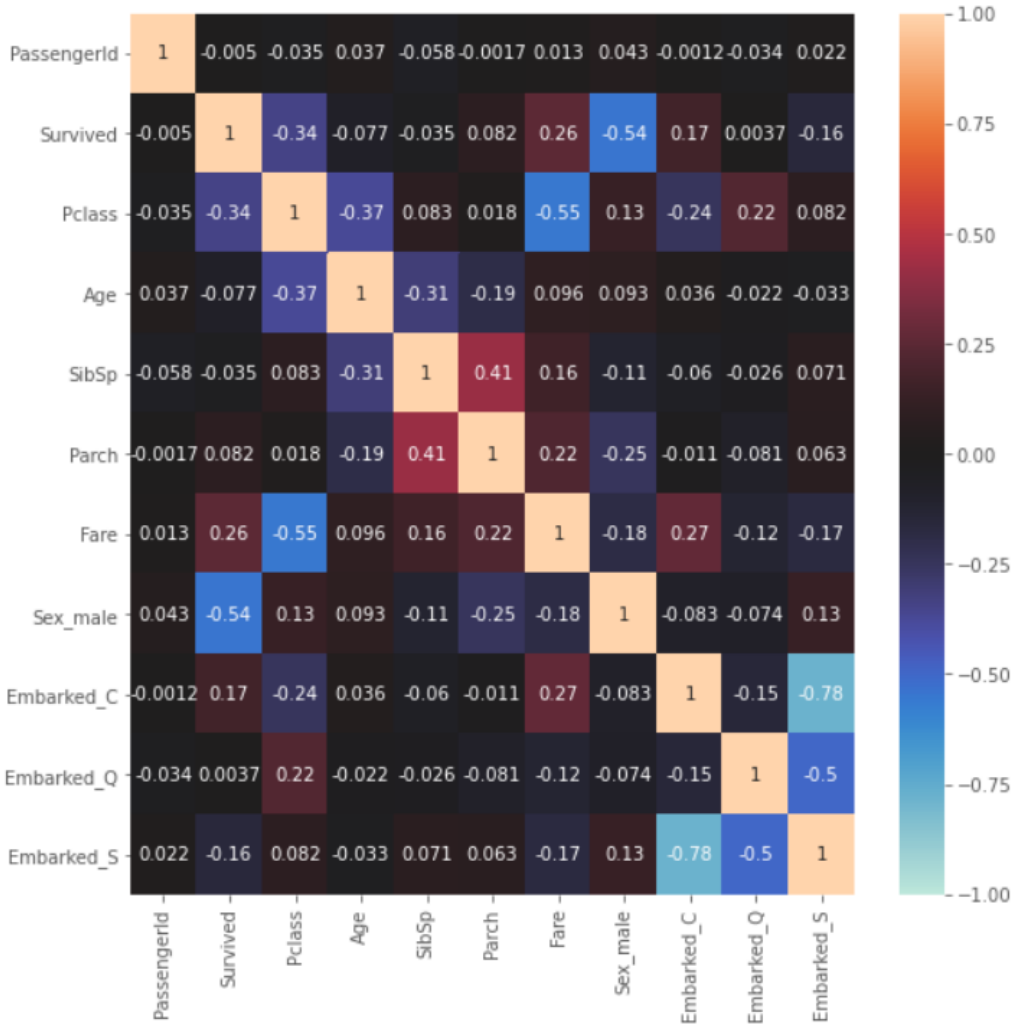
  ▼ 원-핫 인코딩이란?

  Re: 카테고리 변수에 대해 1,0 더미 변수화하여 수치화하는 방법
  - inplace = True: 원본데이터 변경
  - df.corr(): 상관계수를 포함한 상관행렬 작성
  - 히트맵: 데이터의 상대적인 크기를 색상으로 나타내는 시각화 기법

▼ 관련 코드 및 히트맵

```
train_df_corr = pd.get_dummies(train_df, columns=["Sex"],drop_first=True)
train_df_corr = pd.get_dummies(train_df_corr, columns=["Embarked"])

train_corr = train_df_corr.corr() #상관행렬


plt.figure(figsize=(9, 9))
sns.heatmap(train_corr, vmax=1, vmin=-1, center=0, annot=True) #히트맵 시각화
```

- 히트맵



- 상관관계가 가장 높은 변수: Sex_male
- 남성의 낮은 생존율과 여성의 높은 생존율 확인 가능
- 티켓 등급이 높을수록 생존율이 높음
- 연령과 생존은 관련이 낮음

---

4. 데이터 전처리 및 특징값 생성

- **데이터 전처리란?**
  - 분석 및 처리에 적합한 형태로 만드는 과정. 분석 시, 80%의 시간을 사용
- **데이터 전처리 과정**
  - 데이터 수집 → 정제 → 통합 및 축소 → 변환
- **데이터 전처리 방법**

  1) 결측치 처리
  - 수치형인 경우, 평균이나 중앙값으로 대체
  - 범주형인 경우, 분포에서 빈도수가 많은 값으로 대체
  - 예측 모델(KNN...)로 대체

  2) 이상치 처리
  - 표준 점수로 변환 후, -/+3 제거
  - IQR 방식 사용(cf: 박스플롯)
  - 도메인 지식 이용 or Binning 처리

  3) 데이터 단위 변환
  - Standard Scaling: 평균이 0, 분산이 1인 분포로 변환
  - MinMax Scaling: 특정 범위(0~1)로 모든 데이터 변환
  - Box-Cox: 여러 k 값 중 가장 작은 SSE 선택

- **데이터 통합 및 결측치 처리**
  - pd.concat(['데이터프레임1','데이터프레임2'])
  - sort = False 설정은 결합 후 행 순서 변화x
  - reset_index(drop=True)는 결합 후 데이터 행 번호를 삭제
  - pd.merge(데이터프레임1, 데이터프레임2, on = 기준, how = 어디쪽으로)

▼ 관련 코드

```
all_df = pd.concat([train_df, test_df],sort=False).reset_index(drop=True)
all_df.isnull().sum()#전체 결측치 확인
Fare_mean = all_df[["Pclass","Fare"]].groupby("Pclass").mean().reset_index()
Fare_mean.columns = ["Pclass","Fare_mean"]
#Pclass 별 Fare의 평균을 구한 후, Pclass의 평균값으로 결측치를 채움

all_df = pd.merge(all_df, Fare_mean, on="Pclass",how="left")
all_df.loc[(all_df["Fare"].isnull()), "Fare"] = all_df["Fare_mean"]
#이후 Fare를 Fare_mean값으로 변경 (loc 사용)

all_df = all_df.drop("Fare_mean",axis=1) #Fare_mean 값은 이제 불필요하므로 삭제
```

- **호칭에 주목**
  - 호칭: Master, Mr, Miss, Mrs

▼ 관련 코드

- 호칭을 변수로 추가
  - split()를 사용해 ,나 .로 구분한 후, 0부터 세기 시작해 2번째에 나오는 요소가 호칭
  - ex) Braund, Mr. Owen Harris

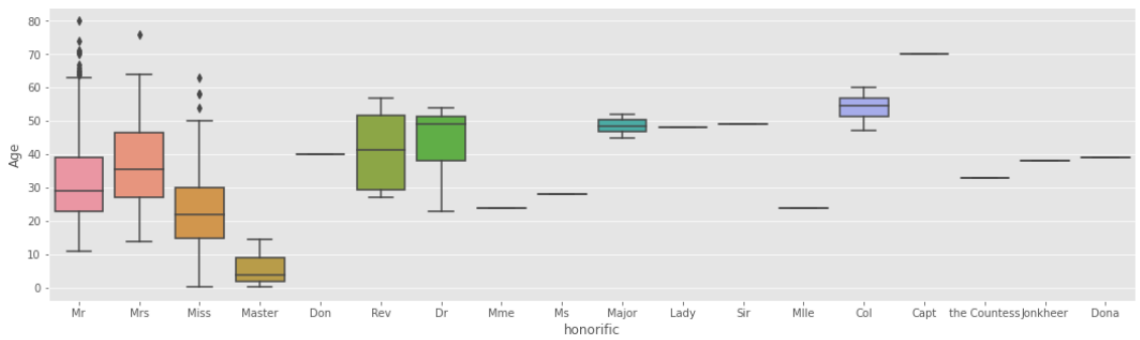
```
name_df = all_df["Name"].str.split("[,.]",n = 2,expand=True)
name_df.columns = ["family_name","honorific","name"]

#공백 삭제
name_df["family_name"] =name_df["family_name"].str.strip()
name_df["honorific"] =name_df["honorific"].str.strip()
name_df["name"] =name_df["name"].str.strip()
```

- 호칭별 연령 분포 확인
  - pd.concat(axis = 0): 세로 결합
  - pd.concat(axis = 1): 가로 결합

```
all_df = pd.concat([all_df, name_df],axis=1)
```

- 박스 플롯으로 시각화
  - sns.boxplot(x=데이터값, y=데이터값, data=이용할 데이터)

```
plt.figure(figsize=(18, 5))
sns.boxplot(x="honorific", y="Age", data=all_df)
```

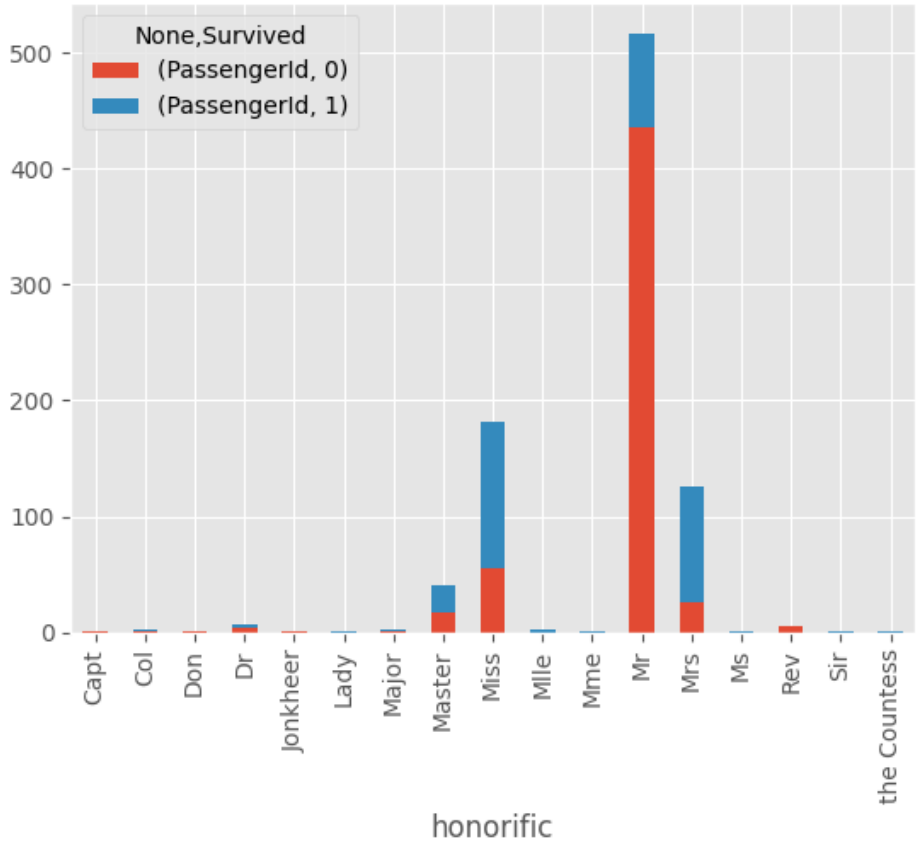


호칭에 걸맞는 연령대의 분포 확인 가능

- 호칭별 생존율 차이 확인

```
train_df = pd.concat([train_df,name_df[0:len(train_df)].reset_index(drop=True)],axis=1)
test_df = pd.concat([test_df,name_df[len(train_df):].reset_index(drop=True)],axis=1)

#호핑과 생존 별로 인원수 집계
honorific_df = train_df[["honorific","Survived","PassengerId"]].dropna().groupby(["honorific","Survived"]).cou
honorific_df.plot.bar(stacked=True)
```

- 호칭별 생존율 차이 확인



- Miss나 Mrs의 생존율이 높고, Mr의 생존율이 낮음
- 남성 호칭 중에서도 Master는 Mr보다 생존율이 높음

- 연령의 결측치를 호칭별 평균 연령으로 보완

```
honorific_age_mean = all_df[["honorific","Age"]].groupby("honorific").mean().reset_index()
honorific_age_mean.columns = ["honorific","honorific_Age"]

all_df = pd.merge(all_df, honorific_age_mean, on="honorific", how="left")
all_df.loc[(all_df["Age"].isnull()), "Age"] = all_df["honorific_Age"]
all_df = all_df.drop(["honorific_Age"],axis=1) #honorific_Age는 불필요하므로 삭제
```

- **가족 인원수에 주목**

▼ 관련 코드

```
all_df["family_num"] = all_df["Parch"] + all_df["SibSp"]
#'동승 중인 부모 및 자녀수'와 '동승 중인 형제 및 배우자 수'를 합하여 가족 인원수 컬럼 생성

all_df.loc[all_df["family_num"] ==0, "alone"] = 1
#홀로 탑승했는지(1), 동반 가족이 있었는지(0) 여부를 추가
all_df["alone"].fillna(0, inplace=True)

all_df = all_df.drop(["PassengerId","Name","family_name","name","Ticket","Cabin"]
,axis=1)
categories = all_df.columns[all_df.dtypes == "object"]
all_df.loc[~((all_df["honorific"] =="Mr") |
    (all_df["honorific"] =="Miss") |
    (all_df["honorific"] =="Mrs") |
    (all_df["honorific"] =="Master")), "honorific"] = "other"
#불필요한 변수 삭제, 수치로 변환, 소수 호칭을 other로 변환
```

- **라벨 인코딩 및 머신러닝 라이브러리 설치**
  - 라벨 인코딩: 각 카테고리명을 임의의 숫자로 대체, 변수의 수는 그대로 유지하면서 수치화가 가능

  ▼ 관련 코드

  - 머신러닝 라이브러리 설치 및 불러오기

```
from sklearn.preprocessing import LabelEncoder
#결측치가 있으며 라벨링이 되지 않으므로, missing값으로 대체
all_df["Embarked"].fillna("missing", inplace=True)

#categories = all_df.columns[all_df.dtypes == "object"]
#Index(['Sex', 'Embarked', 'honorific'], dtype='object')
```

```
for cat in categories:
    le = LabelEncoder()
    print(cat)
    if all_df[cat].dtypes == "object":
        le = le.fit(all_df[cat])
        all_df[cat] = le.transform(all_df[cat])
```

- 모든 데이터를 원래 학습데이터와 테스트 데이터로 돌려놓기

```
train_X = all_df[~all_df["Survived"].isnull()].drop("Survived",axis=1)
.reset_index(drop=True)
train_Y = train_df["Survived"]
test_X = all_df[all_df["Survived"].isnull()].drop("Survived",axis=1)
.reset_index(drop=True)
#Survived 값이 null이 아닐 땐, 학습 데이터로 null일 땐 테스트 데이터로 돌려놓기
```

▶ 과제 할당

- 3.8장 ~ 3.10장
  - 이론 공부 및 코드 실습 후 발표 준비
- fifa.csv 데이터로 각자 1주차(3/16) 전처리 실습

특이 사항

- 다음주부터 영등포역으로 장소 변경
- 첫 스터디임에도 불구하고 다들 발표 준비를 열심히 해오셔서 좋은 자극을 많이 받았습니다~

비고

- 없음
