#Creating combined dataframe

x\_train = train\_data.drop(['Survived', 'Died'], axis=1)

y\_train = train\_data['Survived']

x\_test = test\_data

#cobining the dataframes

df\_combined = x\_train.append(x\_test)

df\_combined.shape

train\_data = pd.read\_csv("/kaggle/input/titanic/train.csv")

test\_data = pd.read\_csv("/kaggle/input/titanic/test.csv")

train\_data["Age"] = train\_data["Age"].fillna(train\_data.describe()["Age"]["mean"])

test\_data["Age"] = test\_data["Age"].fillna(test\_data.describe()["Age"]["mean"])

train\_data['Age']=train\_data['Age'].astype(np.float64)

train\_data["Fare"] = train\_data["Fare"].fillna(train\_data.describe()["Fare"]["mean"])

test\_data["Fare"] = test\_data["Fare"].fillna(test\_data.describe()["Fare"]["mean"])

train\_data['Fare']=train\_data['Fare'].astype(int)

train\_data.head(5)

#women survival

**women = train\_data[train\_data.Sex == 'female']["Survived"]**

#men survival

**men = train\_data[train\_data.Sex == 'male']["Survived"]**

print("Survival rate for women is {:.2f} and for men is {:.2f}".format((sum(women)/len(women))\*100, (sum(men)/len(men))\*100))

train\_data = pd.read\_csv("/kaggle/input/titanic/train.csv")

data = train\_data

data['Died'] = 1 - data['Survived']

fig = go.Figure(data=[

go.Bar(name='Survived', x=data['Sex'], y=[data[data.Sex==data['Sex'][0]]['Survived'].agg(sum), data[data.Sex==data['Sex'][1]]['Survived'].agg(sum)], marker\_color='#EB89B5', opacity=0.75),

go.Bar(name='Died', x=data['Sex'], y=[data[data.Sex==data['Sex'][0]]['Died'].agg(sum), data[data.Sex==data['Sex'][1]]['Died'].agg(sum)], marker\_color='#330C73', opacity=0.75)

])

fig.update\_layout(barmode='stack', width=600, title\_text='Survivors categorized as Male/Female', xaxis\_title\_text='Sex', yaxis\_title\_text='Number of passangers')

fig.show()

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

facet\_data = data[['Sex', 'Survived', 'Embarked', 'Pclass', 'Fare']].groupby(['Sex', 'Survived', 'Embarked', 'Pclass']).agg('sum').reset\_index()

fig = px.bar(facet\_data, x='Sex', y='Fare', facet\_row='Survived', facet\_col='Pclass', color='Embarked', barmode='group')

fig.update\_layout(title\_text='Different view of Passenger survival with respect to Fare, Embarked and Pclass')

fig.show()



fig = go.Figure()

fig.add\_trace(go.Violin(x=data['Sex'][data['Survived']==1], y=data['Age'][data['Survived']==1],

name='Survived', side='negative', legendgroup='Survived', scalegroup='Survived', line\_color='blue'))

fig.add\_trace(go.Violin(x=data['Sex'][data['Died']==1], y=data['Age'][data['Died']==1],

name='Died', side='positive', legendgroup='Died', scalegroup='Died', line\_color='dark orange'))

fig.update\_traces(meanline\_visible=True, width=1)

fig.update\_layout(violingap=0, violinmode='overlay', width=700, title\_text='Survivors categorized as Male/Female related to their Age', xaxis\_title\_text='Sex', yaxis\_title\_text='Age of passangers')

fig.show()

data['Survived'] = data['Survived'].astype(str)

fig\_scatter = px.scatter(data, x='Fare', y='Age', color='Survived', facet\_col='Sex')

fig\_scatter.update\_layout(width=800, title\_text='Survivors categorized as Male/Female', yaxis\_title\_text='Age of passangers')

fig\_scatter.show()

data['Survived'] = data['Survived'].astype(int)

fig\_scatter\_plane = go.Figure()

fig\_scatter\_plane.add\_trace(go.Scatter(x=data[data['Survived']==1]['Age'], y=data[data['Survived']==1]['Fare'],

mode='markers', name='Survived', marker=dict(size=data[data['Survived']==1]['Fare']/8)))

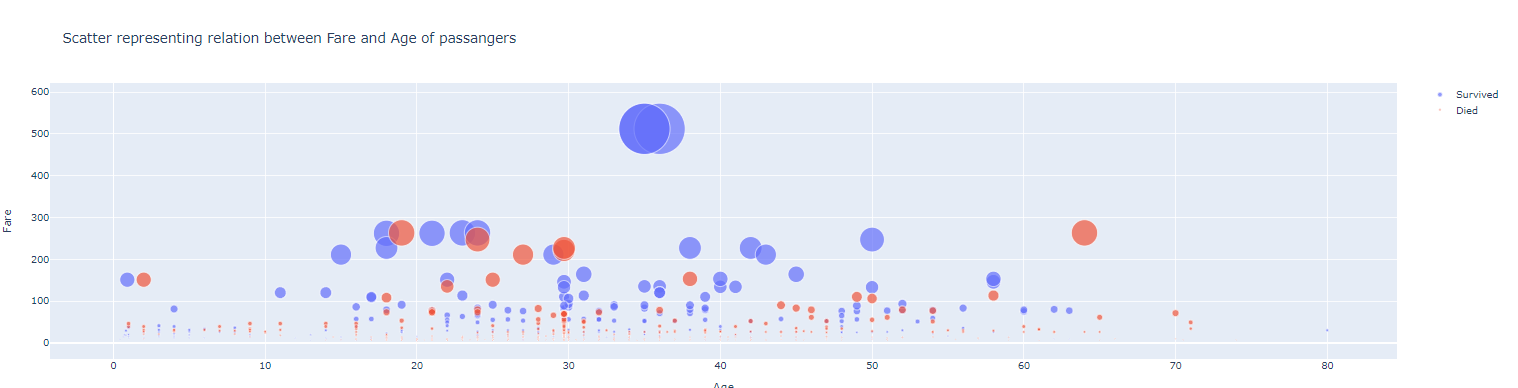
fig\_scatter\_plane.add\_trace(go.Scatter(x=data[data['Died']==1]['Age'], y=data[data['Died']==1]['Fare'],

mode='markers', name='Died', marker=dict(size=data[data['Died']==1]['Fare']/8)))

fig\_scatter\_plane.update\_layout(title\_text='Scatter representing relation between Fare and Age of passangers',

xaxis\_title\_text='Age', yaxis\_title\_text='Fare')

fig\_scatter\_plane.show()



#Creating combined dataframe

x\_train = train\_data.drop(['Survived', 'Died'], axis=1)

y\_train = train\_data['Survived']

x\_test = test\_data

#cobining the dataframes

df\_combined = x\_train.append(x\_test)

df\_combined.shape

#creating funtion for creating categories based on family size

def family\_size():

global df\_combined

df\_combined['FamilySize'] = df\_combined['Parch'] + df\_combined['SibSp'] + 1

#feature to decide family size

df\_combined['Singleton'] = df\_combined['FamilySize'].map(lambda s: 1 if s == 1 else 0)

df\_combined['SmallFamily'] = df\_combined['FamilySize'].map(lambda s: 1 if 2 <= s <= 4 else 0)

df\_combined['LargeFamily'] = df\_combined['FamilySize'].map(lambda s: 1 if 5 <= s <= 8 else 0)

return df\_combined

df\_combined = family\_size()

df\_combined.head()

#Separating the train and test dataframes

x\_train = df\_combined[:891].copy()

x\_test = df\_combined[891:].copy()

x\_test.reset\_index(inplace=True, drop=True)

x\_train.shape, x\_test.shape

random\_forest = RandomForestClassifier()

random\_forest.fit(x\_train, y\_train)

y\_pred\_random\_forest = random\_forest.predict(x\_test)

random\_forest\_accuracy = round(random\_forest.score(x\_train, y\_train)\*100, 2)

log\_regres = LogisticRegression()

log\_regres.fit(x\_train, y\_train)

y\_pred\_log\_regres = log\_regres.predict(x\_test)

log\_regres\_accuracy = round(log\_regres.score(x\_train, y\_train)\*100, 2)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(x\_train, y\_train)

y\_pred\_knn = knn.predict(x\_test)

knn\_accuracy = round(knn.score(x\_train, y\_train)\*100, 2)

gaussian = GaussianNB()

gaussian.fit(x\_train, y\_train)

y\_pred\_gaussian = gaussian.predict(x\_test)

gaussian\_accuracy = round(gaussian.score(x\_train, y\_train)\*100, 2)

perceptron = Perceptron()

perceptron.fit(x\_train, y\_train)

y\_pred\_perceptron = perceptron.predict(x\_test)

perceptron\_accuracy = round(perceptron.score(x\_train, y\_train)\*100, 2)

tree = DecisionTreeClassifier()

tree.fit(x\_train, y\_train)

y\_pred\_tree = tree.predict(x\_train)

tree\_accuracy = round(tree.score(x\_train, y\_train)\*100, 2)

train\_df[['Pclass', 'Survived']].groupby(['Pclass'], as\_index=False).mean().sort\_values(by='Survived', ascending=False)

Out[9]:

train\_df[["Sex", "Survived"]].groupby(['Sex'], as\_index=False).mean().sort\_values(by='Survived', ascending=False)

Out[10]:

train\_df[["SibSp", "Survived"]].groupby(['SibSp'], as\_index=False).mean().sort\_values(by='Survived', ascending=False)

Out[11]:

*# grid = sns.FacetGrid(train\_df, col='Pclass', hue='Survived')*

grid = sns.FacetGrid(train\_df, col='Survived', row='Pclass', size=2.2, aspect=1.6)

grid.map(plt.hist, 'Age', alpha=.5, bins=20)

grid.add\_legend();

*# grid = sns.FacetGrid(train\_df, col='Embarked', hue='Survived', palette={0: 'k', 1: 'w'})*

grid = sns.FacetGrid(train\_df, row='Embarked', col='Survived', size=2.2, aspect=1.6)

grid.map(sns.barplot, 'Sex', 'Fare', alpha=.5, ci=None)

grid.add\_legend()

X\_train = train\_df.drop("Survived", axis=1)

Y\_train = train\_df["Survived"]

X\_test = test\_df.drop("PassengerId", axis=1).copy()

X\_train.shape, Y\_train.shape, X\_test.shape

*# Logistic Regression*

logreg = LogisticRegression()

logreg.fit(X\_train, Y\_train)

Y\_pred = logreg.predict(X\_test)

acc\_log = round(logreg.score(X\_train, Y\_train) \* 100, 2)

acc\_log

coeff\_df = pd.DataFrame(train\_df.columns.delete(0))

coeff\_df.columns = ['Feature']

coeff\_df["Correlation"] = pd.Series(logreg.coef\_[0])

coeff\_df.sort\_values(by='Correlation', ascending=False)

*# Support Vector Machines*

svc = SVC()

svc.fit(X\_train, Y\_train)

Y\_pred = svc.predict(X\_test)

acc\_svc = round(svc.score(X\_train, Y\_train) \* 100, 2)

acc\_svc

knn = KNeighborsClassifier(n\_neighbors = 3)

knn.fit(X\_train, Y\_train)

Y\_pred = knn.predict(X\_test)

acc\_knn = round(knn.score(X\_train, Y\_train) \* 100, 2)

acc\_knn

*# Gaussian Naive Bayes*

gaussian = GaussianNB()

gaussian.fit(X\_train, Y\_train)

Y\_pred = gaussian.predict(X\_test)

acc\_gaussian = round(gaussian.score(X\_train, Y\_train) \* 100, 2)

acc\_gaussian

*# Perceptron*

perceptron = Perceptron()

perceptron.fit(X\_train, Y\_train)

Y\_pred = perceptron.predict(X\_test)

acc\_perceptron = round(perceptron.score(X\_train, Y\_train) \* 100, 2)

acc\_perceptron

*# Linear SVC*

linear\_svc = LinearSVC()

linear\_svc.fit(X\_train, Y\_train)

Y\_pred = linear\_svc.predict(X\_test)

acc\_linear\_svc = round(linear\_svc.score(X\_train, Y\_train) \* 100, 2)

acc\_linear\_svc

*# Decision Tree*

decision\_tree = DecisionTreeClassifier()

decision\_tree.fit(X\_train, Y\_train)

Y\_pred = decision\_tree.predict(X\_test)

acc\_decision\_tree = round(decision\_tree.score(X\_train, Y\_train) \* 100, 2)

acc\_decision\_tree

*# Random Forest*

random\_forest = RandomForestClassifier(n\_estimators=100)

random\_forest.fit(X\_train, Y\_train)

Y\_pred = random\_forest.predict(X\_test)

random\_forest.score(X\_train, Y\_train)

acc\_random\_forest = round(random\_forest.score(X\_train, Y\_train) \* 100, 2)

acc\_random\_forest

for dataset **in** combine:

dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train\_df['Title'], train\_df['Sex'])

for dataset **in** combine:

dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col',\

'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')

dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')

dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train\_df[['Title', 'Survived']].groupby(['Title'], as\_index=False).mean()

title\_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

for dataset **in** combine:

dataset['Title'] = dataset['Title'].map(title\_mapping)

dataset['Title'] = dataset['Title'].fillna(0)

train\_df.head()

train\_df = train\_df.drop(['Name', 'PassengerId'], axis=1)

test\_df = test\_df.drop(['Name'], axis=1)

combine = [train\_df, test\_df]

train\_df.shape, test\_df.shape

for dataset **in** combine:

dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

train\_df.head()

*# grid = sns.FacetGrid(train\_df, col='Pclass', hue='Gender')*

grid = sns.FacetGrid(train\_df, row='Pclass', col='Sex', size=2.2, aspect=1.6)

grid.map(plt.hist, 'Age', alpha=.5, bins=20)

grid.add\_legend()

for dataset **in** combine:

for i **in** range(0, 2):

for j **in** range(0, 3):

guess\_df = dataset[(dataset['Sex'] == i) & \

(dataset['Pclass'] == j+1)]['Age'].dropna()

*# age\_mean = guess\_df.mean()*

*# age\_std = guess\_df.std()*

*# age\_guess = rnd.uniform(age\_mean - age\_std, age\_mean + age\_std)*

age\_guess = guess\_df.median()

*# Convert random age float to nearest .5 age*

guess\_ages[i,j] = int( age\_guess/0.5 + 0.5 ) \* 0.5

for i **in** range(0, 2):

for j **in** range(0, 3):

dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.Pclass == j+1),\

'Age'] = guess\_ages[i,j]

dataset['Age'] = dataset['Age'].astype(int)

train\_df.head()

train\_df['AgeBand'] = pd.cut(train\_df['Age'], 5)

train\_df[['AgeBand', 'Survived']].groupby(['AgeBand'], as\_index=False).mean().sort\_values(by='AgeBand', ascending=True)

for dataset **in** combine:

dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0

dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1

dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2

dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3

dataset.loc[ dataset['Age'] > 64, 'Age']

train\_df.head()

train\_df = train\_df.drop(['AgeBand'], axis=1)

combine = [train\_df, test\_df]

train\_df.head()

for dataset **in** combine:

dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train\_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as\_index=False).mean().sort\_values(by='Survived', ascending=False)

for dataset **in** combine:

dataset['IsAlone'] = 0

dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

train\_df[['IsAlone', 'Survived']].groupby(['IsAlone'], as\_index=False).mean()

train\_df = train\_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)

test\_df = test\_df.drop(['Parch', 'SibSp', 'FamilySize'], axis=1)

combine = [train\_df, test\_df]

train\_df.head()

for dataset **in** combine:

dataset['Age\*Class'] = dataset.Age \* dataset.Pclass

train\_df.loc[:, ['Age\*Class', 'Age', 'Pclass']].head(10)

for dataset **in** combine:

dataset['Embarked'] = dataset['Embarked'].fillna(freq\_port)

train\_df[['Embarked', 'Survived']].groupby(['Embarked'], as\_index=False).mean().sort\_values(by='Survived', ascending=False)

for dataset **in** combine:

dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

train\_df.head()

test\_df['Fare'].fillna(test\_df['Fare'].dropna().median(), inplace=True)

test\_df.head()

train\_df['FareBand'] = pd.qcut(train\_df['Fare'], 4)

train\_df[['FareBand', 'Survived']].groupby(['FareBand'], as\_index=False).mean().sort\_values(by='FareBand', ascending=True)

for dataset **in** combine:

dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0

dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1

dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare'] = 2

dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3

dataset['Fare'] = dataset['Fare'].astype(int)

train\_df = train\_df.drop(['FareBand'], axis=1)

combine = [train\_df, test\_df]

train\_df.head(10)