Vibhu Vanjari Module 5 Assignment 1 – Titanic

Code for this assignment can be found here: LINK1 LINK2

Improving on EDA and missing data:

I wasn't completely happy with the results from last week's assignment, so I wanted to revisit the EDA. I particularly wanted to improve how I imputed missing Age data and I also wanted to improve Ticket Groupings. In addition to looking at what the alphanumeric prefixes to ticket numbers were I also looked at Fare to see how different Ticket Groups fit into Fare buckets. I noticed that the long tail of Ticket Groups with only a few passengers fit into the $7-15 range. Similarly there were Ticket Groups in the $60-70 range and a third set of groups in the $20-40 range.

To improve my age predictions, I ran a random forest model instead of a regression. The random forest model yielded a much lower RMSE on known age data than the regression model. However, this is expected in a small dataset, the RF is likely overfit. I chose not to dig deeper here because either model would be limited by the host of categorical variables. I followed the same methodology: 1) predict missing age data 2) use predicted age data to update Sibling vs Spouse and Parents vs Children split of SibSp and Parch variables and 3) use these updated Sibling, Spouse, Parents and Children data to re-predict missing age. I then imputed the dataset to reflect the new prediction.

Models:

Once I had my data cleaned up, I ran RandomForestClassified, GradientBoostingClassifier and ExtraTreesClassifier models. I first manually conducted hyperparameter tuning adjusting the max_depth, min_samples_split, min_samples_leaf and n_estimator.

However, I switched to using grid_search for doing this. The general idea was creating a param_grid with different values for the different tuning parameters and then cycling through each combination to find the best parameters based on cross-validated accuracy. The GridSearchCV function allows you to implement 5-fold cross-validation within the model to determine this accuracy. While the methodology was quite exciting I was again disappointed with the results. Different models and different hyperparameters didn't seem to significantly improve my results.

While building GB Tree models, I wanted to revaluate the features I was accounting for, after running a variable importance plot and correlation matrix across all features I decided to remove a few. This actually made the model worse, which was quite surprising. A takeaway here is that unlike regressions tree models are quite good at handling correlated variables. My Extra Tree models performed the best, so I decided to run different criteria models using Extra Trees, I also did this experiment on my RF models to compare the two. Changing the criteria had little to no effect on the actual prediction accuracy. I made sure to use a max_depth of 20 when cycling through criteria as this is likely they would have the most significant impact.

All my models (including earlier regressions) to predict Survived scored between 0.73 and 0.79. This makes me believe there are more hidden features within the dataset to uncover. My best model barely outperforms predicting all women survive (0.76555), which is quite surprising. There is likely a simpler way to tackle this dataset than tree-based models.

RF_gini_predictions.csv Complete · 15h ago · RF: (n_estimators=100, max_depth = 5, min_samples_split = 10, min_samples_leaf = 10, random_state=42, criterion='gini')	0.78229
ET_gini_predictions.csv Complete · 15h ago · ET-gini: { 'criterion': 'gini', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100, 'random...	0.78468
ET_predictions.csv Complete · 16h ago · Extra Trees: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}	0.78468
GB_predictions (1).csv Complete · 18h ago · Gradient Boosted Tree: {'learning_rate': 0.01, 'max_depth': 5, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 300}	0.77033
GB_predictions.csv Complete · 19h ago · Gradient Boosted Tree: {'learning_rate': 0.05, 'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}	0.77751
RF_predictions.csv Complete · 19h ago · RF: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}	0.77990
RF_n100_predictions (1).csv Complete · 20h ago · RF with n = 100, depth = 5, min samples split/leaf = 5	0.78229
RF_n100_predictions.csv Complete · 20h ago · RF with n = 100	0.73205

Introduction

Link to access this code - https://colab.research.google.com/drive/1ojYP9ur_Mpa1GtjdkkPL4PsNwrxLD_ir?usp=sharing

Data taken from - https://www.kaggle.com/c/titanic/data

Part 2: https://colab.research.google.com/drive/1OX0dFDrMRPJfsf1d39qfpUsUGJkhzUt-?usp=sharing

Import modules and data files

[ ] ↳ 5 cells hidden

EDA

Establishing base survival rates across different features

[ ] ↳ 4 cells hidden

Cabin

TAKEAWAY: There is not much information the variable "Cabin" gives us in addition to Pclass. The deck level doesn't or room number doesn't neccessarily give us much information on survival rate. There is a lot of missing samples here so it may be best to eliminate this variable.

Other EDA

↳ 7 cells hidden

Merge and clean data

Merge data

```
# Restoring original train data.
df_train_original = pd.read_csv('train.csv')

# Create working datasets
df_train = df_train_original
df_test = df_test_original

# Add a variable to both df_train and df_test indicating whether train or test dataset
df_train['TestYes'] = 0
df_test['TestYes'] = 1

# Create merged dataset used to cleanup data
df_merged = pd.concat([df_train, df_test], ignore_index=True)


df_merged.info()
```

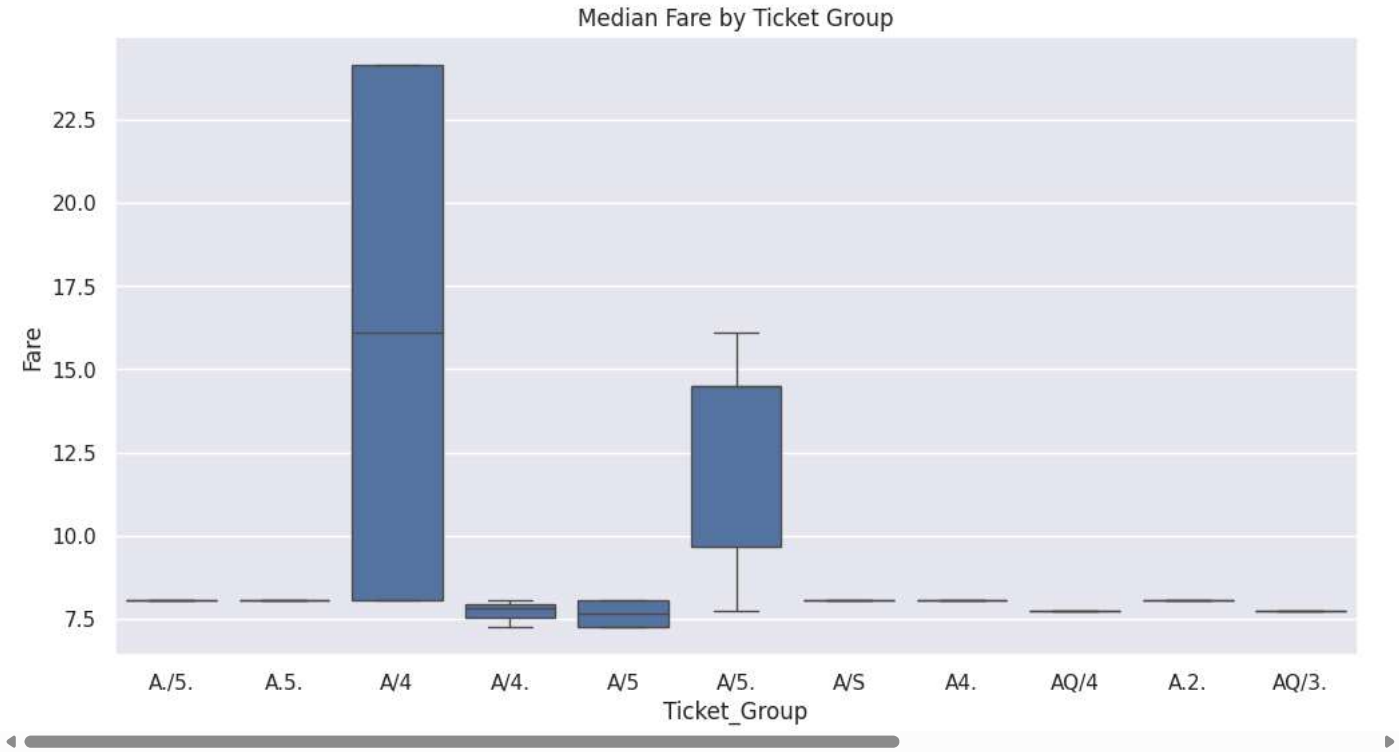
Show hidden output

Creating a variable for ticket type

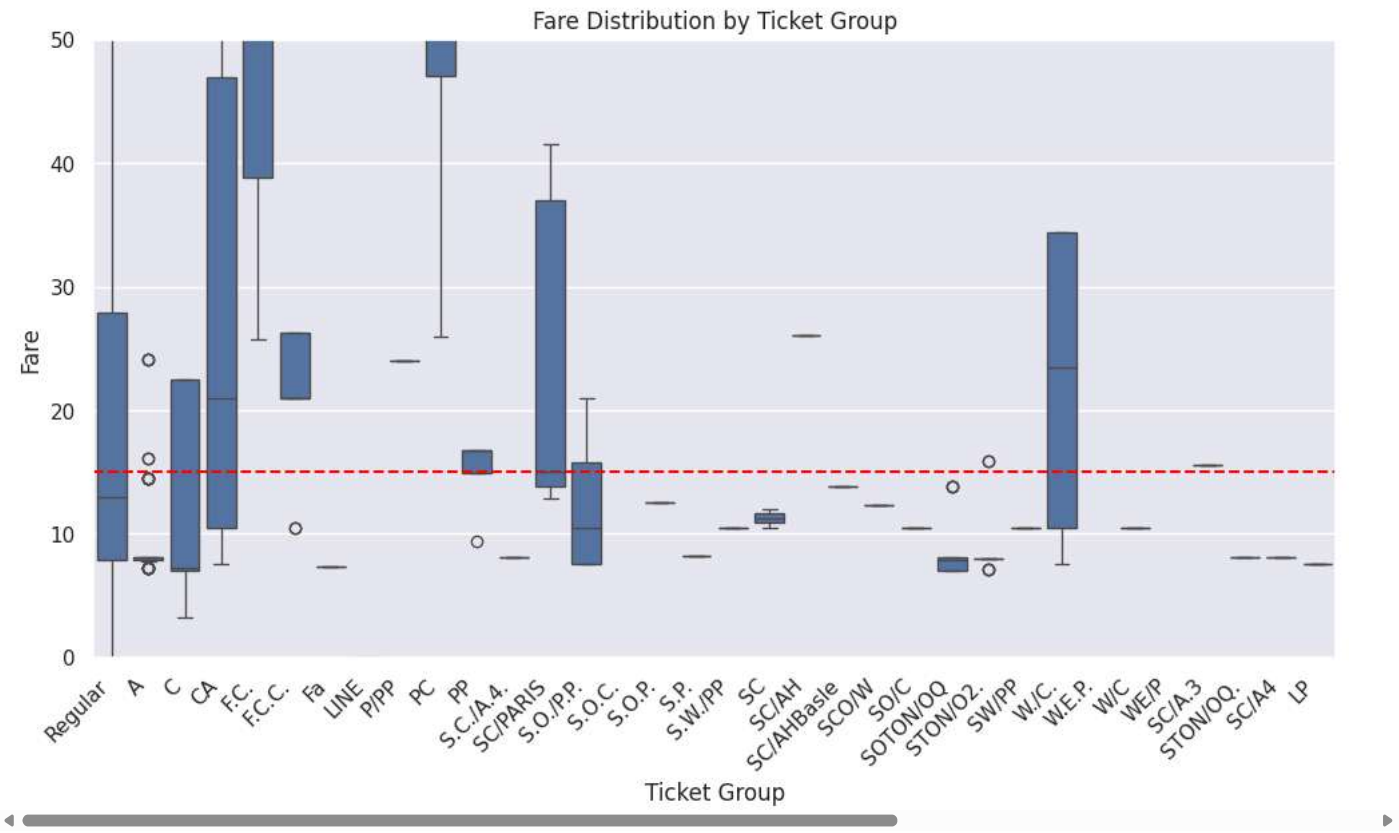
```
# Split numeric text into a seperate Ticket feature
df_merged['Ticket_Number'] = df_merged['Ticket'].str.extract(r'(\d+)$')
df_merged['Ticket_Group'] = df_merged['Ticket'].str.replace(r'\d+$', '', regex=True)
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].str.replace(' ', '', regex=False)


# If 'Ticket_Number' is N/A set to 0
df_merged['Ticket_Number'] = df_merged['Ticket_Number'].fillna(0)

# If 'Ticket_Group' is " " set to "Regular"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(r'^\s*$', 'Regular', regex=True)
```

```
# Plot median fares of all ticket_group that start with "A"
plt.figure(figsize=(12, 6))
sns.boxplot(x='Ticket_Group', y='Fare', data=df_merged[df_merged['Ticket_Group'].str.startswith('A')])
plt.title('Median Fare by Ticket Group')
```

```
Text(0.5, 1.0, 'Median Fare by Ticket Group')
```




```
# Merge all Ticket_Group that start with "A"
df_merged.loc[df_merged['Ticket_Group'].str.startswith('A', na=False), 'Ticket_Group'] = 'A'

# Merge Ticket_Group "C.A.", "CA.", "CA", "C.A./SOTON" into "CA"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['C.A.', 'CA.', 'CA', 'C.A./SOTON'], 'CA')

# Merge Ticket_Group "SOTON/O.Q.", "SOTON/O2" and "SOTON/OQ" into "SOTON/OQ"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['SOTON/O.Q.', 'SOTON/OQ', 'SOTON/O2'], 'SOTON/OQ')

# Merge SC/PARIS, SC/Paris, S.C./PARIS → SC/PARIS
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['SC/PARIS', 'SC/Paris', 'S.C./PARIS'], 'SC/PARIS')


# Merge all other Ticket_Group values into "OTHER"
# other_ticket_groups = df_merged['Ticket_Group'].unique()
# other_ticket_groups = other_ticket_groups[~pd.Series(other_ticket_groups).isin(['A/5', 'A/4', 'CA', 'SOTON/OQ', 'Regular', 'STON/O2.', 'SC
# df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(other_ticket_groups, 'OTHER')



# Plot fare vs ticket type
plt.figure(figsize=(12, 6))
sns.boxplot(x='Ticket_Group', y='Fare', data=df_merged)
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels by 45 degrees
plt.title('Fare Distribution by Ticket Group') # Adding a title for better clarity
plt.xlabel('Ticket Group')
plt.ylabel('Fare')
plt.ylim(0,50)
plt.axhline(y=15, color='red', linestyle='--', label='Fare = 15')
plt.show()
```




```
# Merge other ticket groups with fare under 20
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace([
    'LP', 'SC/A4', 'STON/OQ.', 'W/C', 'SW/PP', 'STON/O2.', 'SOTON/OQ', 'SO/C',
    'SCO/W', 'SC', 'S.W./PP', 'S.P.', 'S.O.P.', 'S.O./P.P.', 'S.C./A.4.', 'Fa',
    'SC/AHBasle', 'SC/A.3', 'PP', 'C'
], 'Other_under20')


# Count by Ticket_Group
df_merged['Ticket_Group'].value_counts()
```

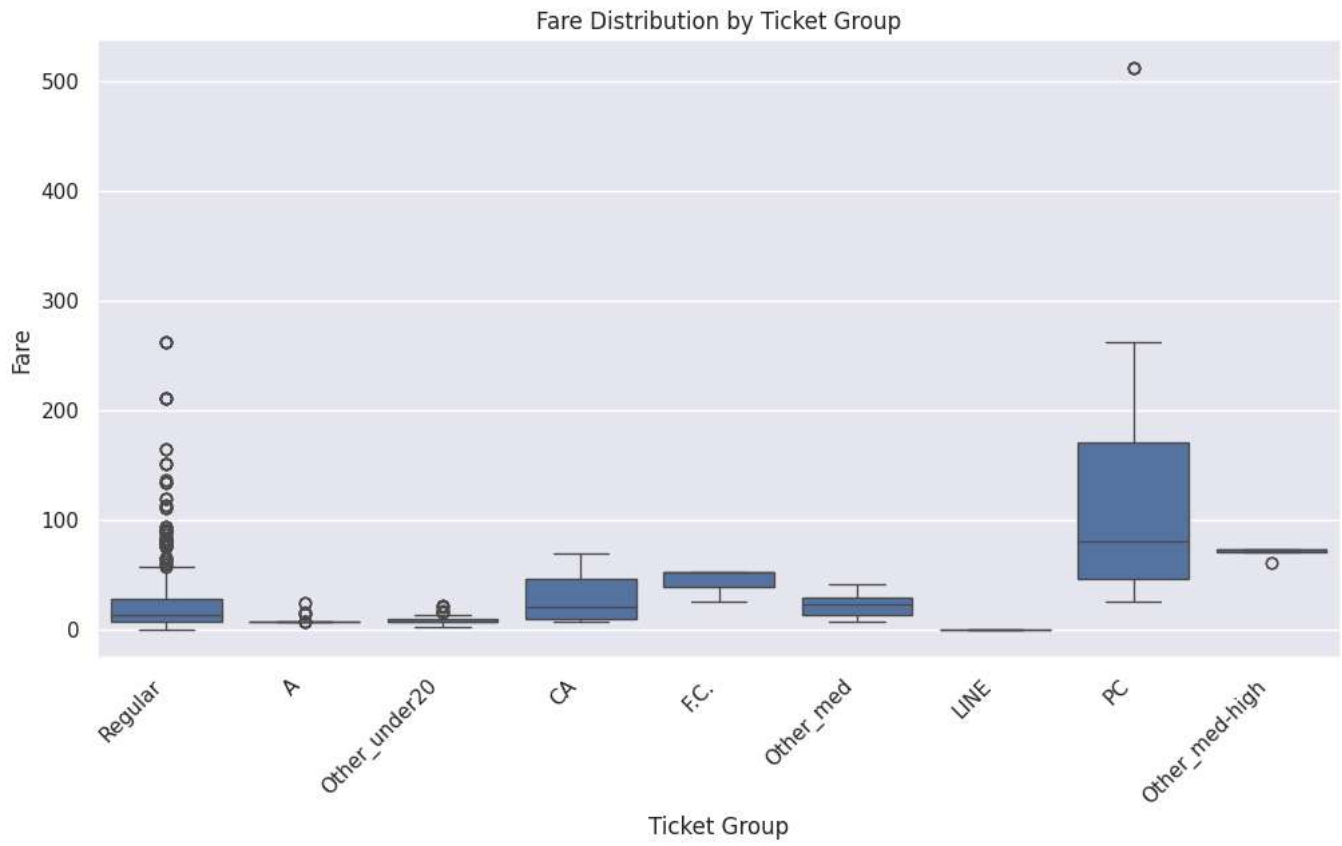
Ticket_Group	count
Regular	957
PC	92
Other_under20	83
CA	69
A	42
SC/PARIS	19
W./C.	14
F.C.C.	9
S.O.C.	7
SC/AH	4
LINE	4
F.C.	3
P/PP	2
W.E.P.	2
WE/P	2

dtype: int64

```
# Describe fare for all Ticket Groups
df_merged.groupby('Ticket_Group')['Fare'].describe().sort_values(by='mean', ascending=False)
```

Ticket_Group	count	mean	std	min	25%	50%	75%	max
PC	92.0	126.336726	113.567462	25.9250	47.028150	80.6854	170.541675	512.3292
S.O.C.	7.0	73.500000	0.000000	73.5000	73.500000	73.5000	73.500000	73.5000
WE/P	2.0	71.000000	0.000000	71.0000	71.000000	71.0000	71.000000	71.0000
W.E.P.	2.0	61.175000	0.000000	61.1750	61.175000	61.1750	61.175000	61.1750
F.C.	3.0	43.247233	15.160237	25.7417	38.870850	52.0000	52.000000	52.0000
CA	69.0	29.983877	20.975517	7.5500	10.500000	21.0000	46.900000	69.5500
Regular	956.0	27.810364	37.887928	0.0000	7.885425	13.0000	27.900000	263.0000
SC/AH	4.0	26.000000	0.000000	26.0000	26.000000	26.0000	26.000000	26.0000
SC/PARIS	19.0	24.892105	12.379018	12.8750	13.860400	15.0500	37.004200	41.5792
P/PP	2.0	24.000000	0.000000	24.0000	24.000000	24.0000	24.000000	24.0000
W./C.	14.0	22.516071	10.770997	7.5500	10.500000	23.4500	34.375000	34.3750
F.C.C.	9.0	20.416667	6.125000	10.5000	21.000000	21.0000	26.250000	26.2500
A	42.0	10.189681	4.757010	7.2500	7.862500	8.0500	8.050000	24.1500
Other_under20	83.0	9.776255	4.121077	3.1708	7.281250	7.9250	10.500000	22.5250
LINE	4.0	0.000000	0.000000	0.0000	0.000000	0.0000	0.000000	0.0000

```
# Merge ticket groups "S.O.C.", "W.E.P." and "WE/P"
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['S.O.C.', 'W.E.P.', 'WE/P'], 'Other_med-high')

# Merge ticket groups "SC/AH", "SC/PARIS", "P/PP", "W./C.", "F.C.C."
df_merged['Ticket_Group'] = df_merged['Ticket_Group'].replace(['SC/AH', 'SC/PARIS', 'P/PP', 'W./C.', 'F.C.C.'], 'Other_med')


# Plot fare vs ticket type
plt.figure(figsize=(12, 6))
sns.boxplot(x='Ticket_Group', y='Fare', data=df_merged)
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels by 45 degrees
plt.title('Fare Distribution by Ticket Group') # Adding a title for better clarity
plt.xlabel('Ticket Group')
plt.ylabel('Fare')
plt.show()
```



Creating a variable for Title

```
# Look for "Mr.", "Master.", "Mrs.", "Miss." in the Name feature and extract to a new feature called "Title"
df_merged['Title'] = df_merged['Name'].str.extract(r' ([A-Za-z]+)\.', expand=False)
df_merged['Title'] = df_merged['Title'].str.replace(' ', '', regex=False)

# Count by Title
df_merged['Title'].value_counts()
```

Title	count
Mr	757
Miss	260
Mrs	197
Master	61
Dr	8
Rev	8
Col	4
Major	2
Ms	2
Mlle	2
Countess	1
Jonkheer	1
Lady	1
Mme	1
Sir	1
Don	1
Capt	1
Dona	1

dtype: int64

```
# Standardize titles
df_merged['Title'] = df_merged['Title'].replace({
    'Mlle': 'Miss',
    'Ms': 'Miss',
    'Mme': 'Mrs',
    'Dr': 'Officer',
    'Rev': 'Officer',
    'Col': 'Officer',
    'Major': 'Officer',
    'Capt': 'Officer',
    'Sir': 'Royalty',
    'Lady': 'Royalty',
    'Countess': 'Royalty',
    'Don': 'Royalty',
    'Dona': 'Royalty',
    'Jonkheer': 'Royalty'
})

# Count by Title
df_merged['Title'].value_counts()
```

Title	count
Mr	757
Miss	264
Mrs	198
Master	61
Officer	23
Royalty	6

dtype: int64

Splitting SibSp into two variables

```
# Create variables Siblings, Spouse
df_merged['Siblings'] = 0
df_merged['Spouse'] = 0

# If Title is Master, Miss or Age < 18 then Siblings = SibSp (this is a child or unmarried person)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] < 18)), 'Siblings'] = df_merged['SibSp']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] < 18)), 'Spouse'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age >= 18, SibSp > 0, then Spouse = 1
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] >= 18) & (df_merged['SibSp'] > 0)), 'Spous
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] >= 18) & (df_merged['SibSp'] > 0)), 'Sibli
```

Splitting Parch into two variables

```
# Create variables Siblings, Spouse
df_merged['Parents'] = 0
df_merged['Children'] = 0

# If Title is Master, Miss or Age <= 20 then Parents = Parch (this is a child/young adult traveling with parents)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] <= 20)), 'Parents'] = df_merged['Parch']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age'] <= 20)), 'Children'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age > 20, Parch > 0 then Children = Parch
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] > 20) & (df_merged['Parch'] > 0)), 'Childr
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age'] > 20) & (df_merged['Parch'] > 0)), 'Parent
```

Predicting missing Embarked data

[ ] ↳ 8 cells hidden

Predicting missing Fare data

```
# Filter to missing Fare data
missing_fare = df_merged[df_merged['Fare'].isnull()]

# Print rows
print(missing_fare)
```

```
      PassengerId  Survived  Pclass                Name   Sex   Age  SibSp  \
1043         1044       NaN       3  Storey, Mr. Thomas  male  60.5      0

      Parch Ticket  Fare Cabin Embarked  TestYes Ticket_Number Ticket_Group  \
1043      0   3701   NaN   NaN        S        1          3701      Regular

     Title  Siblings  Spouse  Parents  Children
1043    Mr         0       0        0         0
```

```
# Filter to other passengers in 3rd class, embarking from Southampton
filtered_df = df_merged[(df_merged['Pclass'] == 3) & (df_merged['Embarked'] == 'S')]

# Calculate fare statistics
fare_stats = filtered_df['Fare'].describe()
fare_stats['median'] = filtered_df['Fare'].median()

# Print the results
print(fare_stats)
```

```
count     494.000000
mean       14.435422
std        13.118281
min         0.000000
25%         7.854200
50%         8.050000
75%        15.900000
max        69.550000
median      8.050000
Name: Fare, dtype: float64
```

Missing fare for this passenger can be set to the median fare for 3rd class passengers embarking from Southampton.

Clean data

[ ] ↳ 4 cells hidden

Predicting age

Comparing regression vs RF model for predicting Age

```
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   PassengerId        1309 non-null   int64
 1   Survived           891 non-null    float64
 2   Pclass             1309 non-null   int64
 3   Name               1309 non-null   object
 4   Sex                1309 non-null   object
 5   Age                1046 non-null   float64
 6   SibSp              1309 non-null   int64
 7   Parch              1309 non-null   int64
 8   Ticket             1309 non-null   object
 9   Fare               1309 non-null   float64
 10  Embarked           1309 non-null   object
 11  TestYes            1309 non-null   int64
 12  Ticket_Number      1309 non-null   object
 13  Ticket_Group       1309 non-null   object
 14  Title              1309 non-null   object
 15  Siblings           1309 non-null   int64
 16  Spouse             1309 non-null   int64
 17  Parents            1309 non-null   int64
 18  Children           1309 non-null   int64
 19  Sex_code           1309 non-null   int8
 20  Embarked_code      1309 non-null   int8
 21  Ticket_Group_code  1309 non-null   int8
 22  Title_Code         1309 non-null   int8
dtypes: float64(3), int64(9), int8(4), object(7)
memory usage: 199.5+ KB
```

```
# Regression3: trial and error to get lowest p-values, however Parents and Children variables cannot be used to predict missing age values
age_model3 = smf.ols(formula='Age ~ C(Pclass) + Parents + Children + C(Embarked_code) + C(Title_Code)', data=df_merged).fit()
print(age_model3.summary())
```

Show hidden output

```
# Random forest to predict age
df_age_rf = df_merged[df_merged['Age'].notnull()].copy()

# Select features
features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Siblings', 'Spouse', 'Parents', 'Children', 'Sex_code', 'Embarked_code', 'Ticket_Grc
X = df_age_rf[features]
y = df_age_rf['Age']

# Train model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X, y)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

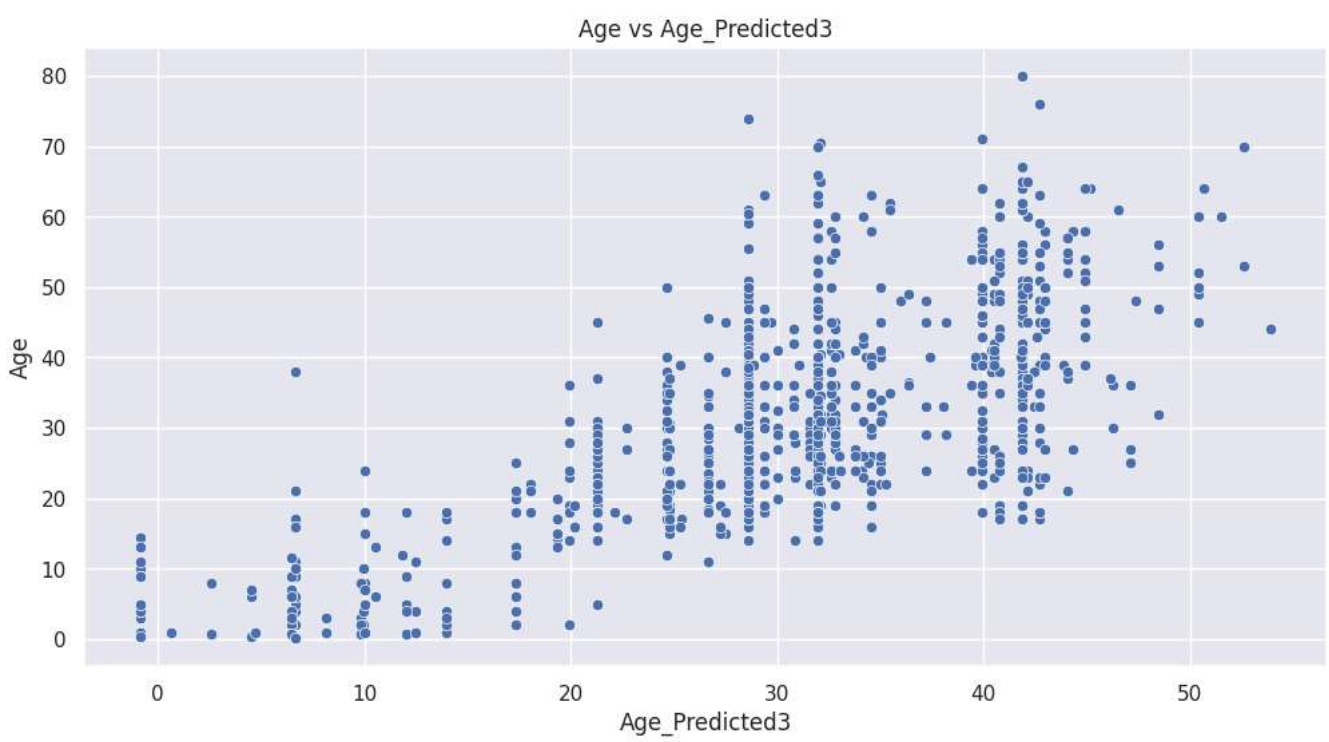
```
# Make age predictions for missing data using regression
df_merged['Age_Predicted3'] = age_model3.predict(df_merged)


# Make age predictions for missing data using random forest
df_merged['Age_Predicted_RF'] = rf_model.predict(df_merged[features])


# Mask where Age is known (to use as ground truth)
mask = df_merged['Age'].notnull()
```

```
# Calculate RMSE for both models
rmse_RF = np.sqrt(mean_squared_error(df_merged.loc[mask, 'Age'], df_merged.loc[mask, 'Age_Predicted_RF']))
rmse_reg3 = np.sqrt(mean_squared_error(df_merged.loc[mask, 'Age'], df_merged.loc[mask, 'Age_Predicted3']))

print(f"RMSE for Random Forest: {rmse_RF}")
print(f"RMSE for Regression3: {rmse_reg3}")
```

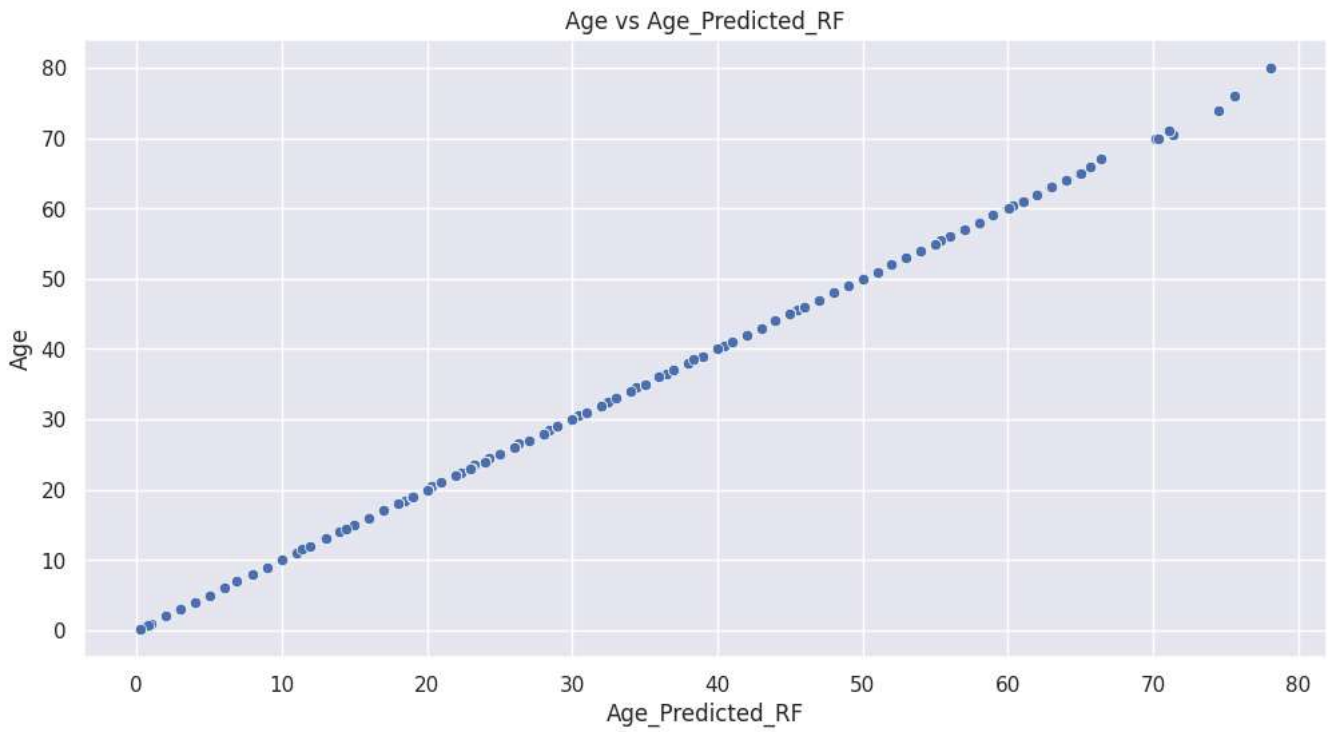
```
RMSE for Random Forest: 0.07645610038974587
RMSE for Regression3: 10.409408119948672
```

```
# Download df_merged as csv
df_merged.to_csv('df_merged.csv', index=False)
```

```
# Plot Age_Predicted3 vs Age for samples where Age is not missing
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Age_Predicted3', y='Age', data=df_merged[~df_merged['Age'].isnull()])
plt.title('Age vs Age_Predicted3')
plt.xlabel('Age_Predicted3')
plt.ylabel('Age')
plt.show()
```




```
# Plot Age_Predicted_RF vs Age for samples where Age is not missing
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Age_Predicted_RF', y='Age', data=df_merged[~df_merged['Age'].isnull()])
plt.title('Age vs Age_Predicted_RF')
plt.xlabel('Age_Predicted_RF')
plt.ylabel('Age')
plt.show()
```



Updating Siblings, Spouse, Parents and Children Using Preliminary Predictions

```
# If Title is Master, Miss or Age < 18 then Siblings = SibSp (this is a child or unmarried person)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted3'] < 18)), 'Siblings'] = df_merged['SibSp']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted3'] < 18)), 'Spouse'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age >= 18, SibSp > 0, then Spouse = 1
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted3'] >= 18) & (df_merged['SibSp'] >
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted3'] >= 18) & (df_merged['SibSp'] >

# If Title is Master, Miss or Age <= 20 then Parents = Parch (this is a child/young adult traveling with parents)
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted3'] <= 20)), 'Parents'] = df_merged['Parch']
df_merged.loc[((df_merged['Title'].isin(['Master', 'Miss'])) | (df_merged['Age_Predicted3'] <= 20)), 'Children'] = 0

# If Title is Mr, Mrs, Royalty or Officer and Age > 20, Parch > 0 then Children = Parch
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted3'] > 20) & (df_merged['Parch'] > 0
df_merged.loc[((df_merged['Title'].isin(['Mr', 'Mrs', 'Royalty', 'Officer'])) & (df_merged['Age_Predicted3'] > 20) & (df_merged['Parch'] > 0
```

Rerunning Best Age Prediction

```
# Random forest to predict age
df_age_rf = df_merged[df_merged['Age'].notnull()].copy()

# Select features
features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Siblings', 'Spouse', 'Parents', 'Children', 'Sex_code', 'Embarked_code', 'Ticket_Grou
X = df_age_rf[features]
y = df_age_rf['Age']

# Train model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X, y)

# Make age predictions for missing data using random forest
df_merged['Age_Predicted_RF'] = rf_model.predict(df_merged[features])


# Plot Age_Predicted3 vs Age for samples where Age is not missing
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Age_Predicted_RF', y='Age', data=df_merged[~df_merged['Age'].isnull()]
plt.title('Age vs Age_Predicted_RF')
plt.xlabel('Age_Predicted_RF')
```

```
plt.ylabel('Age')
```

Introduction

Link to access this code - https://colab.research.google.com/drive/1OX0dFDrMRPJfsf1d39qfpUsUGJkhzUt-#scrollTo=y4rZOE0FzvrZ

Data taken from - https://www.kaggle.com/c/titanic/data

Part 1: https://colab.research.google.com/drive/1ojYP9ur_Mpa1GtjdkkPL4PsNwrxLD_ir?usp=sharing

Import modules and data files

[ ] ↳ 3 cells hidden

Split Train and Test data

↳ 3 cells hidden

Initial random Forest Regressor Model

Score: 0.78229

```
# Select features
features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Siblings', 'Spouse', 'Parents','Children', 'Sex_code', 'Embarked_code', 'Ticket_Grou
X = df_train[features]
y = df_train['Survived']

# Split data into 80% train, 20% validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


# Random forest to predict Survived using training dataset

# Train model
rf_model = RandomForestClassifier(n_estimators=100, max_depth = 5, min_samples_split = 10, min_samples_leaf = 10, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on validation set
y_val_pred = rf_model.predict(X_val)

# Evaluate performance
accuracy = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {accuracy:.3f}")
print(classification_report(y_val, y_val_pred))
```

```
Validation Accuracy: 0.799
                precision    recall  f1-score   support

           0.0       0.79      0.91      0.85       110
           1.0       0.81      0.62      0.70        69

      accuracy                           0.80       179
     macro avg       0.80      0.77      0.78       179
  weighted avg       0.80      0.80      0.79       179
```

```
y_train_pred = rf_model.predict(X_train)
train_acc = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_acc:.3f}")
```

```
Training Accuracy: 0.857
```

Grid search for best random forest

Score: 0.77990

```
# Define parameter grid
param_grid = {
    'n_estimators': [100],
    'max_depth': [4, 5, 6, 7],
```

```
    'min_samples_split': [5, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}

# Initialize model
rf = RandomForestClassifier(random_state=42)

# Set up GridSearchCV
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,                    # 5-fold cross-validation
    n_jobs=-1,               # Use all CPU cores
    verbose=1
)

# Run grid search on training data
grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validated Accuracy:", grid_search.best_score_)

# Evaluate on validation set using best model
best_rf = grid_search.best_estimator_
y_val_pred = best_rf.predict(X_val)
val_acc = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy (on holdout set): {val_acc:.3f}")

# Evaluate on training set
y_train_pred = best_rf.predict(X_train)
train_acc = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy (on training set): {train_acc:.3f}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
Best Cross-Validated Accuracy: 0.8272825765783512
Validation Accuracy (on holdout set): 0.821
Training Accuracy (on training set): 0.861
```

```
# Make predictions on test data
df_test['Survived_RF_n100'] = rf_model.predict(df_test[features])
df_test['Survived_best_RF'] = best_rf.predict(df_test[features])


# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_best_RF']].to_csv('RF_predictions.csv', index=False)


# Export df_test
df_test.to_csv('df_test.csv', index=False)
```

Gradient Boosted Trees

Score: 0.77751

```
# Define parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [2, 3, 4],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3]
}

# Initialize the model
gb = GradientBoostingClassifier(random_state=42)

# Set up GridSearchCV
grid_search_gb = GridSearchCV(
    estimator=gb,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
```

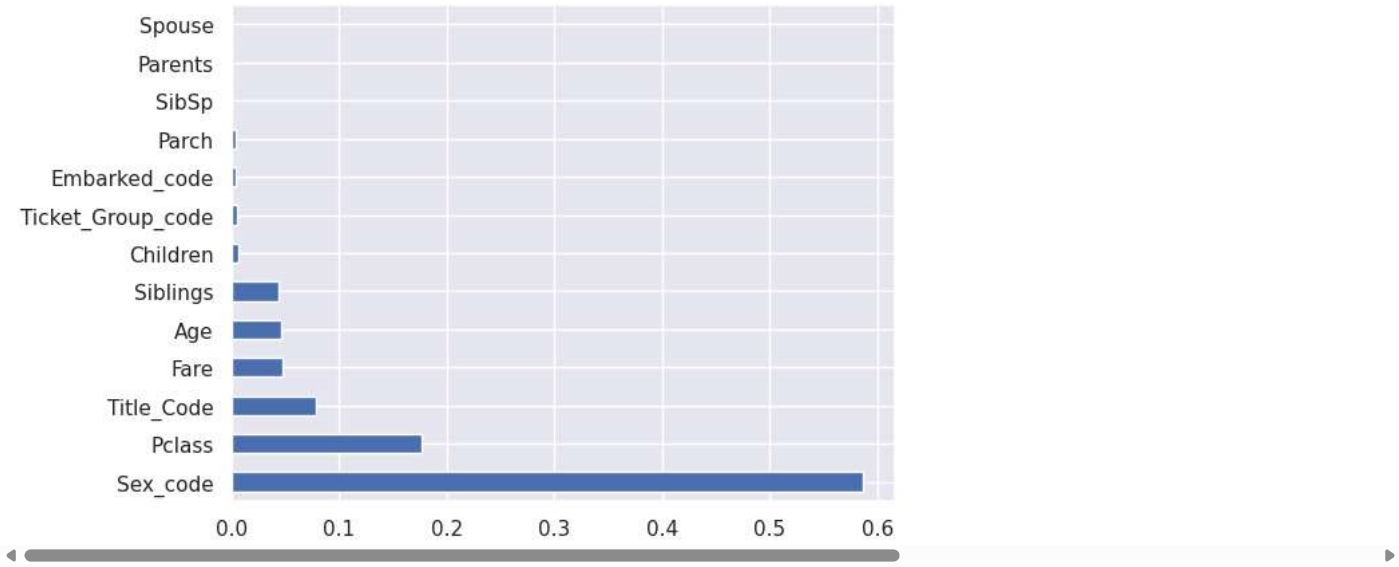
```
    n_jobs=-1,
    verbose=1
)

# Fit on training data
grid_search_gb.fit(X_train, y_train)

# Best model and evaluation
best_gb = grid_search_gb.best_estimator_
print("Best Params:", grid_search_gb.best_params_)
print(f"Best CV Accuracy: {grid_search_gb.best_score_:.3f}")

# Evaluate on validation set
y_val_pred = best_gb.predict(X_val)
val_acc = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy (holdout set): {val_acc:.3f}")
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Params: {'learning_rate': 0.05, 'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best CV Accuracy: 0.836
Validation Accuracy (holdout set): 0.810
```

```
# Make predictions on test data
df_test['Survived_GB'] = best_gb.predict(df_test[features])

# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_GB']].to_csv('GB_predictions.csv', index=False)

# Export df_test
df_test.to_csv('df_test.csv', index=False)
```

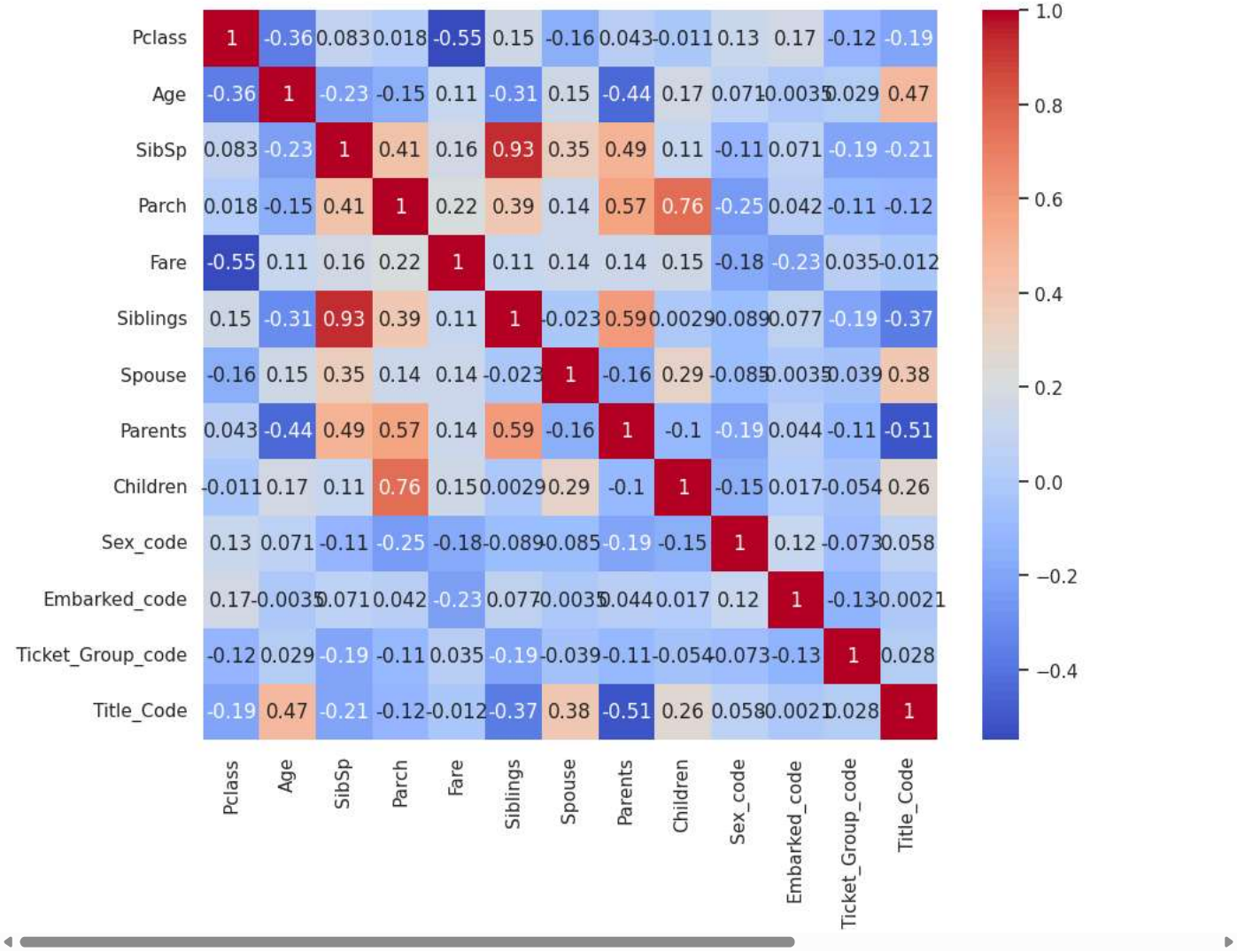
```
importances = pd.Series(best_gb.feature_importances_, index=features)
importances.sort_values(ascending=False).plot(kind='barh')
```

```
<Axes: >
```




```
corr = df_train[features].corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

<Axes: >



Changing features selected - GB Tree 2

Score: 0.77033

Removed variables: Parch, Parents, Spouse

```
# Select features
features = ['Pclass', 'Age', 'SibSp', 'Fare', 'Siblings', 'Children', 'Sex_code', 'Embarked_code', 'Ticket_Group_code', 'Title_Code']
X = df_train[features]
y = df_train['Survived']

# Split data into 80% train, 20% validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


# Define parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [2, 3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5]
}

# Initialize the model
gb = GradientBoostingClassifier(random_state=42)

# Set up GridSearchCV
grid_search_gb = GridSearchCV(
    estimator=gb,
```

```
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)

# Fit on training data
grid_search_gb.fit(X_train, y_train)

# Best model and evaluation
best_gb = grid_search_gb.best_estimator_
print("Best Params:", grid_search_gb.best_params_)
print(f"Best CV Accuracy: {grid_search_gb.best_score_:.3f}")

# Evaluate on validation set
y_val_pred = best_gb.predict(X_val)
val_acc = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy (holdout set): {val_acc:.3f}")
```

```
Fitting 5 folds for each of 432 candidates, totalling 2160 fits
Best Params: {'learning_rate': 0.01, 'max_depth': 5, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 300}
Best CV Accuracy: 0.846
Validation Accuracy (holdout set): 0.838
```

```
# Make predictions on test data
df_test['Survived_GB'] = best_gb.predict(df_test[features])

# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_GB']].to_csv('GB_predictions.csv', index=False)

# Export df_test
df_test.to_csv('df_test.csv', index=False)
```

Extra Trees

Score: 0.78468

```
# Select features
features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Siblings', 'Spouse', 'Parents','Children', 'Sex_code', 'Embarked_code', 'Ticket_Grou
X = df_train[features]
y = df_train['Survived']

# Split data into 80% train, 20% validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None]
}

# Initialize the model
et = ExtraTreesClassifier(random_state=42)

# Set up GridSearchCV
grid_search_et = GridSearchCV(
    estimator=et,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    n_jobs=-1,
    verbose=1
)

# Fit on training data
grid_search_et.fit(X_train, y_train)

# Best model and evaluation
best_et = grid_search_et.best_estimator_
print("Best Params:", grid_search_et.best_params_)
```

```
print(f"Best CV Accuracy: {grid_search_et.best_score_:.3f}")

# Evaluate on validation set
y_val_pred = best_et.predict(X_val)
val_acc = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy (holdout set): {val_acc:.3f}")
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best Params: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}
Best CV Accuracy: 0.833
Validation Accuracy (holdout set): 0.816
```

```
# Make predictions on test data
df_test['Survived_ET'] = best_et.predict(df_test[features])

# Export df_test PassengerId and Survived
df_test[['PassengerId', 'Survived_ET']].to_csv('ET_predictions.csv', index=False)

# Export df_test
df_test.to_csv('df_test.csv', index=False)
```

Experimenting with Gini vs Entropy vs Log-loss on the ET model

Score: 0.78468

```
# Best-found fixed parameters
base_params = {
    'max_depth': 20,
    'max_features': 'sqrt',
    'min_samples_leaf': 2,
    'min_samples_split': 10,
    'n_estimators': 100
}

# Criteria to test
criteria = ['gini', 'entropy', 'log_loss']
results = []

# Evaluate each criterion
for crit in criteria:
    model = ExtraTreesClassifier(criterion=crit, **base_params)

    # Cross-validation accuracy (mean of 5-fold)
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    cv_mean = np.mean(cv_scores)

    # Fit on full training data and evaluate on validation set
    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)
    val_acc = accuracy_score(y_val, y_val_pred)

    results.append((crit, cv_mean, val_acc))

# Print results
for crit, cv_acc, val_acc in results:
    print(f"Criterion: {crit:8s} | CV Accuracy: {cv_acc:.3f} | Validation Accuracy: {val_acc:.3f}")
```

```
Criterion: gini     | CV Accuracy: 0.831 | Validation Accuracy: 0.827
Criterion: entropy  | CV Accuracy: 0.827 | Validation Accuracy: 0.821
Criterion: log_loss | CV Accuracy: 0.827 | Validation Accuracy: 0.821
```

```
gini_params = {
    'criterion': 'gini',
    'max_depth': 20,
    'max_features': 'sqrt',
    'min_samples_leaf': 2,
    'min_samples_split': 10,
    'n_estimators': 100,
    'random_state': 42
}

# Train model on full training set
et_gini = ExtraTreesClassifier(**gini_params)
et_gini.fit(X_train, y_train)
```

```
# Make predictions on test data
df_test['Survived_ET_gini'] = et_gini.predict(df_test[features])

# Export predictions (PassengerId + Survived column)
df_test[['PassengerId', 'Survived_ET']].to_csv('ET_gini_predictions.csv', index=False)
```

Random Forest using Gini criterion

Score: 0.78229

```
# Random forest to predict Survived using training dataset

# Train model
rf_model = RandomForestClassifier(n_estimators=100, max_depth = 5, min_samples_split = 10, min_samples_leaf = 10, random_state=42, criterion=
rf_model.fit(X_train, y_train)

# Predict on validation set
y_val_pred = rf_model.predict(X_val)

# Evaluate performance
accuracy = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {accuracy:.3f}")
print(classification_report(y_val, y_val_pred))

y_train_pred = rf_model.predict(X_train)
train_acc = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_acc:.3f}")
```

```
Validation Accuracy: 0.799
                  precision    recall  f1-score   support

              0.0       0.79      0.91      0.85       110
              1.0       0.81      0.62      0.70        69

         accuracy                           0.80       179
        macro avg       0.80      0.77      0.78       179
     weighted avg       0.80      0.80      0.79       179

     Training Accuracy: 0.857
```

```
# Make predictions on test data
df_test['Survived_RF_gini'] = rf_model.predict(df_test[features])

# Export predictions (PassengerId + Survived column)
df_test[['PassengerId', 'Survived_RF_gini']].to_csv('RF_gini_predictions.csv', index=False)
```