Multi-label classification with complete EDA

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import gc
import re as re
from collections import Counter
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier

import seaborn as sns
from tqdm.auto import tqdm
import math
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, GridSear
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix, Confusior
import warnings
warnings.filterwarnings('ignore')

import time
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
%matplotlib inline
tqdm.pandas()

rc = {
    "axes.facecolor": "#FFF9ED",
    "figure.facecolor": "#FFF9ED",
    "axes.edgecolor": "#000000",
    "grid.color": "#EBEBE7",
    "font.family": "serif",
    "axes.labelcolor": "#000000",
    "xtick.color": "#000000",
    "ytick.color": "#000000",
    "grid.alpha": 0.4
}

sns.set(rc=rc)

from colorama import Style, Fore
red = Style.BRIGHT + Fore.RED
blu = Style.BRIGHT + Fore.BLUE
mgt = Style.BRIGHT + Fore.MAGENTA
gld = Style.BRIGHT + Fore.YELLOW
res = Style.RESET_ALL
```

In [3]:
```
train = pd.read_csv('/input/train.csv')
test = pd.read_csv('/input/test.csv')
```

EDA

In [4]:
```
# summary table function
pd.options.display.float_format = '{:,.2f}'.format
def summary(df):
    print(f'data shape: {df.shape}')
    summ = pd.DataFrame(df.dtypes, columns=['data type'])
    summ['#missing'] = df.isnull().sum().values
    summ['%missing'] = df.isnull().sum().values / len(df) * 100
    summ['#unique'] = df.nunique().values
    desc = pd.DataFrame(df.describe(include='all').transpose())
    summ['min'] = desc['min'].values
    summ['max'] = desc['max'].values
    summ['average'] = desc['mean'].values
    summ['standard_deviation'] = desc['std'].values
    summ['first value'] = df.loc[0].values
    summ['second value'] = df.loc[1].values
    summ['third value'] = df.loc[2].values

    return summ
```

In [5]:
```
summary(train).style.background_gradient(cmap='YlOrBr')
```

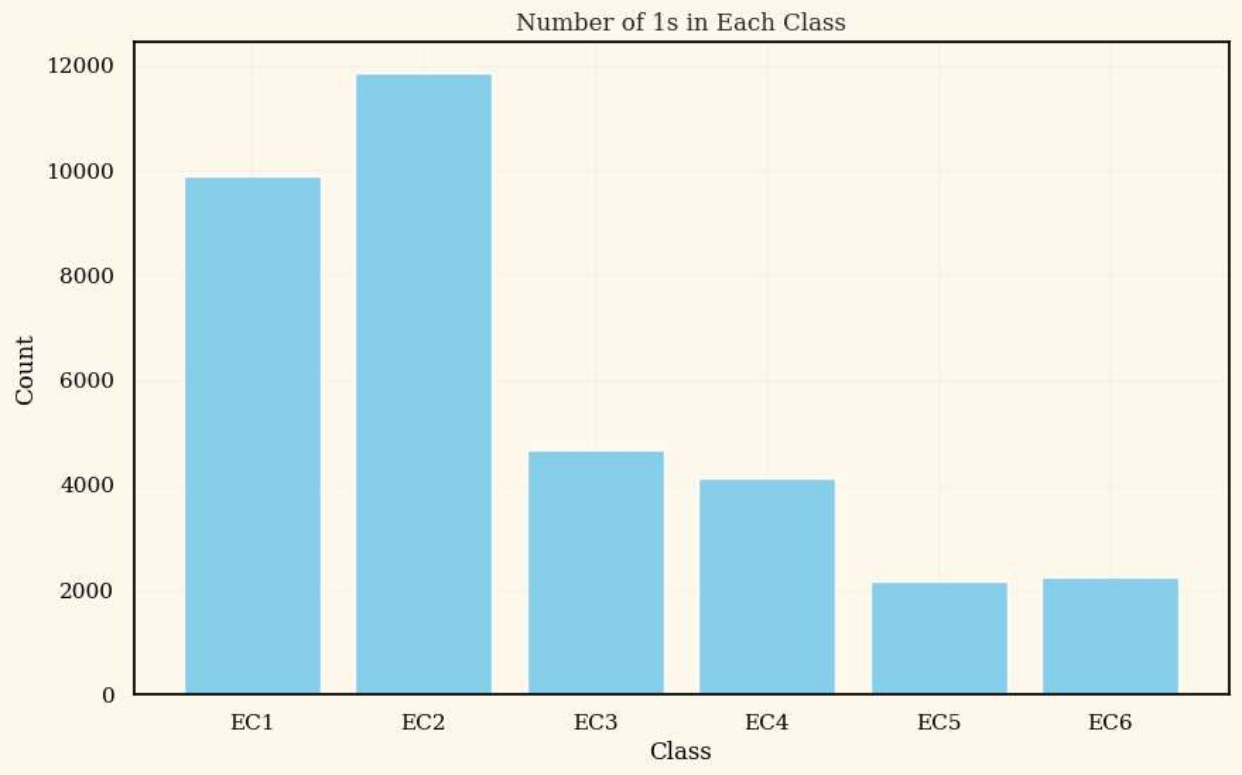
```
data shape: (14838, 38)
```

Out[5]:

	data type	#missing	%missing	#unique	min	max	average	st
id	int64	0	0.000000	14838	0.000000	14837.000000	7418.500000	
BertzCT	float64	0	0.000000	2368	0.000000	4069.959780	515.153604	
Chi1	float64	0	0.000000	1259	0.000000	69.551167	9.135189	
Chi1n	float64	0	0.000000	3157	0.000000	50.174588	5.854307	
Chi1v	float64	0	0.000000	3306	0.000000	53.431954	6.738497	
Chi2n	float64	0	0.000000	3634	0.000000	32.195368	4.432570	
Chi2v	float64	0	0.000000	3725	0.000000	34.579313	5.253221	
Chi3v	float64	0	0.000000	3448	0.000000	22.880836	3.418749	
Chi4n	float64	0	0.000000	2930	0.000000	16.072810	1.773472	
EState_VSA1	float64	0	0.000000	719	0.000000	363.705954	29.202823	
EState_VSA2	float64	0	0.000000	445	0.000000	99.936429	10.435316	
ExactMolWt	float64	0	0.000000	1666	1.007276	2237.318490	292.623087	
FpDensityMorgan1	float64	0	0.000000	556	-666.000000	3.000000	1.236774	
FpDensityMorgan2	float64	0	0.000000	650	-666.000000	3.200000	1.812070	
FpDensityMorgan3	float64	0	0.000000	654	-666.000000	3.400000	2.255470	
HallKierAlpha	float64	0	0.000000	388	-7.730000	0.820000	-1.207776	
HeavyAtomMolWt	float64	0	0.000000	860	0.000000	2035.133000	274.950211	
Kappa3	float64	0	0.000000	2245	-104.040000	1512.242231	5.874372	
MaxAbsEStateIndex	float64	0	0.000000	2356	0.000000	15.630251	10.556443	
MinEStateIndex	float64	0	0.000000	2142	-6.327514	6.000000	-2.119772	
NumHeteroatoms	int64	0	0.000000	40	0.000000	42.000000	8.584108	
PEOE_VSA10	float64	0	0.000000	250	0.000000	97.663462	11.021644	
PEOE_VSA14	float64	0	0.000000	291	0.000000	482.434223	17.790011	
PEOE_VSA6	float64	0	0.000000	219	0.000000	375.425148	8.962440	
PEOE_VSA7	float64	0	0.000000	262	0.000000	211.501279	11.318811	
PEOE_VSA8	float64	0	0.000000	237	0.000000	100.348416	6.704487	
SMR_VSA10	float64	0	0.000000	409	0.000000	80.742293	15.666766	
SMR_VSA5	float64	0	0.000000	492	0.000000	492.729739	31.066423	
SlogP_VSA3	float64	0	0.000000	217	0.000000	115.406157	13.636941	
VSA_EState9	float64	0	0.000000	1946	-5.430556	384.450519	49.309959	
fr_COO	int64	0	0.000000	8	0.000000	8.000000	0.458215	
fr_COO2	int64	0	0.000000	8	0.000000	8.000000	0.459226	
EC1	int64	0	0.000000	2	0.000000	1.000000	0.667745	

	data type	#missing	%missing	#unique	min	max	average	st
EC2	int64	0	0.000000	2	0.000000	1.000000	0.798962	
EC3	int64	0	0.000000	2	0.000000	1.000000	0.313789	
EC4	int64	0	0.000000	2	0.000000	1.000000	0.279081	
EC5	int64	0	0.000000	2	0.000000	1.000000	0.144831	
EC6	int64	0	0.000000	2	0.000000	1.000000	0.151570	

In [6]:
```
# let's check the label variables.

# Define the labels
labels = ['EC1', 'EC2', 'EC3', 'EC4', 'EC5', 'EC6']

# Count the number of 1s in each class
counts = [train[label].sum() for label in labels]

# Create the bar plot
plt.figure(figsize=(10,6))
plt.bar(labels, counts, color='skyblue')
plt.title('Number of 1s in Each Class')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



💡 About Dataset:

- **Data Size**: The dataset contains 14,838 rows and 38 columns.
- **Data Types**: The data contains features with data types int64 and float64.

- **Missing Values**: No column has missing values in the dataset, which is a great sign and simplifies the data cleaning process.
- **Unique Values**: The number of unique values varies greatly among features, with some features like 'id' having unique values for each entry and others like 'EC1', 'EC2', etc., having only two unique values. This suggests a mix of categorical and continuous variables.
- **Statistical Details**: The 'min', 'max', 'average', and 'standard deviation' values indicate the range and dispersion of data for each column, highlighting potential outliers or anomalies.
- **Irrelevant Features**: The 'id' column, as a unique identifier, may not provide meaningful insights for analysis and could potentially be dropped.
- **Multi-label Classification**: This dataset poses a multi-label classification task with 'EC1' through 'EC6' serving as labels. Each of these columns represents a different class to predict, with values of either 0 or 1. Importantly, multiple labels can be true simultaneously. For instance, a data point could have both 'EC1' and 'EC2' as 1, indicating that it belongs to both classes.

This summary offers a comprehensive initial understanding of the dataset's characteristics, helping to inform subsequent steps in your data analysis process.

In [7]:
```
# Column names in the DataFrame
columns = list(train.columns)

# Remove 'EC1' to 'EC6' and 'id'
features = [col for col in columns if col not in ['id', 'EC1', 'EC2', 'EC3', 'EC4', 'E

print(features)
```

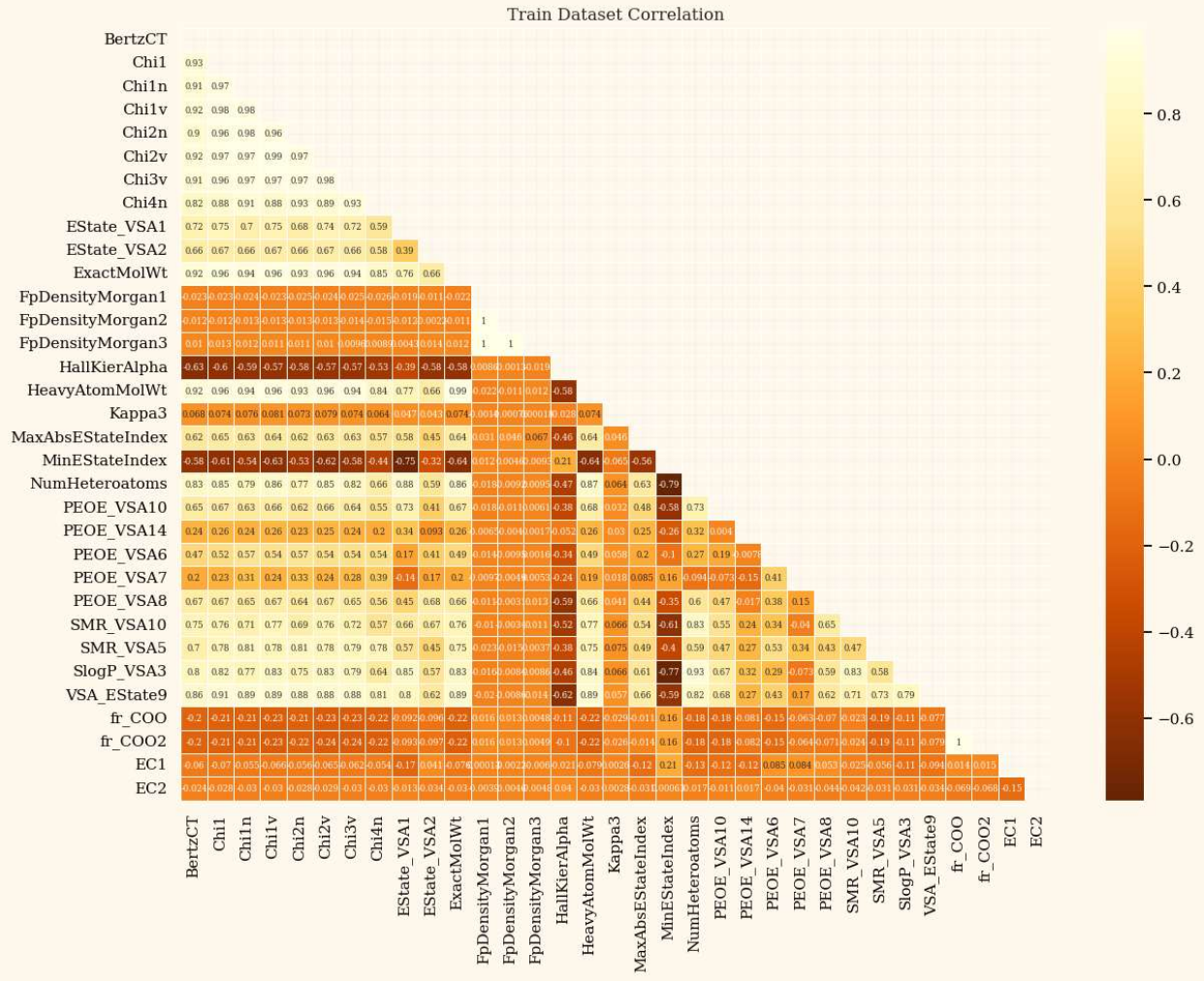
```
['BertzCT', 'Chi1', 'Chi1n', 'Chi1v', 'Chi2n', 'Chi2v', 'Chi3v', 'Chi4n', 'EState_VSA
1', 'EState_VSA2', 'ExactMolWt', 'FpDensityMorgan1', 'FpDensityMorgan2', 'FpDensityMo
rgan3', 'HallKierAlpha', 'HeavyAtomMolWt', 'Kappa3', 'MaxAbsEStateIndex', 'MinEStateI
ndex', 'NumHeteroatoms', 'PEOE_VSA10', 'PEOE_VSA14', 'PEOE_VSA6', 'PEOE_VSA7', 'PEOE_
VSA8', 'SMR_VSA10', 'SMR_VSA5', 'SlogP_VSA3', 'VSA_EState9', 'fr_COO', 'fr_COO2']
```

In [8]:
```
plt.figure(figsize=(14,20))

for idx,column in enumerate(features[:10]):
    # Plotting for 'EC1'
    plt.subplot(10,2,idx*2+1)
    sns.histplot(x=column, hue="EC1", data=train,bins=30,kde=True,palette='YlOrRd')
    plt.title(f"{column} Distribution for EC1")
    plt.ylim(0,100)
    plt.tight_layout()

    # Plotting for 'EC2'
    plt.subplot(10,2,idx*2+2)
    sns.histplot(x=column, hue="EC2", data=train,bins=30,kde=True,palette='YlOrRd')
    plt.title(f"{column} Distribution for EC2")
    plt.ylim(0,100)
    plt.tight_layout()
```

The main point is that if the distributions of 'EC1' and 'EC2' are significantly different, this feature could be useful in predicting the class. Although most of them appear to be similar, it

can be observed that variables like 'EState_VSA1' are quite distinct. Only 10 features were selected and examined due to the large number of features.

In [9]:
```
def plot_correlation_heatmap(df: pd.core.frame.DataFrame, title_name: str = 'Train cor
    # create excluded list
    excluded_columns = ['id', 'EC3', 'EC4', 'EC5', 'EC6']

    # Create a list of columns excluding the excluded_columns.
    columns_without_excluded = [col for col in df.columns if col not in excluded_colum

    corr = df[columns_without_excluded].corr()
    fig, axes = plt.subplots(figsize=(14, 10))
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    sns.heatmap(corr, mask=mask, linewidths=.5, cmap='YlOrBr_r', annot=True, annot_kws
    plt.title(title_name)
    plt.show()

plot_correlation_heatmap(train[columns], 'Train Dataset Correlation')
```



When examining the correlation plot, it becomes evident that many variables exhibit high correlations. Nevertheless, it is crucial to exercise caution since the interpretation of what constitutes a 'high' or 'low' correlation can vary across different domains. It is worth noting that in the context of enzyme data, it is common for variables to display higher correlations. However, it is essential to consider additional perspectives and domain-specific knowledge to

accurately assess the significance of these correlations. Relying solely on correlation values for drawing conclusions may result in misleading interpretations. Therefore, a comprehensive analysis that takes into account various factors is necessary to gain a deeper understanding of the data and make meaningful inferences.

Feature engineering and baseline modeling

About modeling:

The difference between multi-label classification and multi-class classification is as follows:

Multi-label Classification: Multi-label classification deals with scenarios where each data instance can be assigned to multiple labels or classes. This means that a single data point can belong to multiple classes simultaneously. For example, in image classification, an image may contain multiple objects, and the task is to recognize and classify all of these objects simultaneously. This problem is addressed using multi-label classification.

Multi-class Classification: On the other hand, multi-class classification deals with scenarios where each data instance can be assigned to only one class out of several possible classes. Each data point is assigned to a single class exclusively. For example, in digit recognition for handwritten digits, each digit is considered as a separate class, and the task is to classify each digit image into its corresponding class. This problem is addressed using multi-class classification.

In the given problem, where EC1 and EC2 can both be applicable to the same data instance, it is a multi-label problem. This means that each data point can be assigned both the EC1 and EC2 labels simultaneously.

In our case, the dataset presents a multi-label problem where our specific task is to predict the first two features (EC1 and EC2).

As a result, we need to categorize the labels into three distinct categories: EC1, EC2, and others. This classification is necessary because each data point can simultaneously belong to both EC1 and EC2.

By employing the multi-label approach, we can effectively predict the membership of each data point in the EC1, EC2, or other classes.

In [10]:
```
# delete columns from EC3 to EC6 + id
train.drop(['id','EC3', 'EC4', 'EC5', 'EC6'], axis=1, inplace=True)
```

In [11]:
```
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay

def f_importance_plot(f_imp):
    fig = plt.figure(figsize=(12, 0.20*len(f_imp)))
    plt.title('Feature importances', size=16, y=1.05,
              fontweight='bold', color='#444444')
    a = sns.barplot(data=f_imp, x='avg_imp', y='feature',
                    palette='YlOrBr_r', linestyle="-",
                    linewidth=0.5, edgecolor="black")
```

```
    plt.xlabel('')
    plt.xticks([])
    plt.ylabel('')
    plt.yticks(size=11, color='#444444')

    for j in ['right', 'top', 'bottom']:
        a.spines[j].set_visible(False)
    for j in ['left']:
        a.spines[j].set_linewidth(0.5)
    plt.tight_layout()
    plt.show()

def show_confusion_roc(preds: np.array, target: np.array) -> None:
    """Draws a confusion matrix and roc_curve with AUC score.

        Args:
            preds: Predictions from the model.
            target: True labels.

        Returns:
            None
    """

    f, ax = plt.subplots(1, 2, figsize=(13.3, 4))
    df = pd.DataFrame({'preds': preds, 'target': target})
    cm = confusion_matrix(df.target, df.preds.ge(0.5).astype(int))
    cm_display = ConfusionMatrixDisplay(cm).plot(cmap='YlOrBr_r', ax=ax[0])
    ax[0].grid(False)
    RocCurveDisplay.from_predictions(df.target, df.preds, color='#20BEFF', ax=ax[1])
    plt.tight_layout()


def get_mean_auc(oof, target):
    """oof: ['val_idx', 'preds', 'target']"""
    mean_val_auc = roc_auc_score(train[target], oof)
    return mean_val_auc
```

Run the XGBoost (XGB), CatBoost (CatBoost), and LightGBM (LGBM) models with basic hyperparameter settings using k-fold cross-validation. While hyperparameter tuning is necessary to enhance performance, I have created a simple code for the purpose of illustration.

Modeling - XGB & Catboost & LGBM

In [12]:
```
# Target columns
target_cols = ['EC1', 'EC2']

# Features
features = ['BertzCT', 'Chi1', 'Chi1n', 'Chi1v', 'Chi2n', 'Chi2v', 'Chi3v', 'Chi4n',
            'EState_VSA1', 'EState_VSA2', 'ExactMolWt', 'FpDensityMorgan1',
            'FpDensityMorgan2', 'FpDensityMorgan3', 'HallKierAlpha',
            'HeavyAtomMolWt', 'Kappa3', 'MaxAbsEStateIndex', 'MinEStateIndex',
            'NumHeteroatoms', 'PEOE_VSA10', 'PEOE_VSA14', 'PEOE_VSA6', 'PEOE_VSA7',
            'PEOE_VSA8', 'SMR_VSA10', 'SMR_VSA5', 'SlogP_VSA3', 'VSA_EState9',
            'fr_COO', 'fr_COO2']
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

```python
# Create models for each target
models = {
    'xgb': {target: XGBClassifier(n_estimators=1000, n_jobs=-1, max_depth=4, eta=0.2,
    'lgb': {target: LGBMClassifier(n_estimators=1000, n_jobs=-1, max_depth=4, learning
    'cat': {target: CatBoostClassifier(n_estimators=1000, depth=4, learning_rate=0.2,
}

# Dictionaries to store results
oof = {model: {target: [] for target in target_cols} for model in models.keys()}
auc_scores = {model: {target: [] for target in target_cols} for model in models.keys()
best_iters = {model: {target: [] for target in target_cols} for model in models.keys()
best_models = {model: {target: None for target in target_cols} for model in models.key
best_auc = {model: {target: 0 for target in target_cols} for model in models.keys()}

# Training parameters
FOLDS = 5
SEED = 1004
skf = StratifiedKFold(n_splits=FOLDS, shuffle=True, random_state=SEED)

# Train models
for model_name, model_dict in models.items():
    print(f'\033[1;34mTraining {model_name} models\033[0m')  # Blue for model names
    for target in target_cols:
        print(f'\033[1;32mTraining model for {target}\033[0m')  # Green for targets
        y = train[target]
        for fold, (train_idx, val_idx) in enumerate(skf.split(train, y)):
            print(f'\033[1;33m########################## Training FOLD {fold+1} ########
            X_train, y_train = train.iloc[train_idx][features], y.iloc[train_idx]
            X_valid, y_valid = train.iloc[val_idx][features], y.iloc[val_idx]
            model = model_dict[target]
            model.fit(X_train, y_train, eval_set=[(X_valid, y_valid)], early_stopping_
            val_preds = model.predict_proba(X_valid)[:, 1]
            val_score = roc_auc_score(y_valid, val_preds)
            print(f'\033[1;35mauc: {val_score:.5f}\033[0m')  # Purple for AUC scores
            oof[model_name][target].append(val_preds)
            auc_scores[model_name][target].append(val_score)
            if model_name == 'xgb':  # XGBoost
                best_iters[model_name][target].append(model.get_booster().best_ntree_l
            elif model_name == 'lgb':  # LightGBM
                best_iters[model_name][target].append(model.best_iteration_)
            # Save the best model
            if val_score > best_auc[model_name][target]:
                best_auc[model_name][target] = val_score
                best_models[model_name][target] = model
        print(f'\033[1;31m*****************************************\033[0m')  # Re
        print(f'\033[1;35mMean AUC for {target}: {np.mean(auc_scores[model_name][targe
```

```
Training xgb models
Training model for EC1
######################### Training FOLD 1 ########################
auc: 0.70053
######################### Training FOLD 2 ########################
auc: 0.70958
######################### Training FOLD 3 ########################
auc: 0.69603
######################### Training FOLD 4 ########################
auc: 0.69733
######################### Training FOLD 5 ########################
auc: 0.70806
*****************************************
Mean AUC for EC1: 0.70231
Training model for EC2
######################### Training FOLD 1 ########################
auc: 0.55703
######################### Training FOLD 2 ########################
auc: 0.59024
######################### Training FOLD 3 ########################
auc: 0.59280
######################### Training FOLD 4 ########################
auc: 0.57422
######################### Training FOLD 5 ########################
auc: 0.57763
*****************************************
Mean AUC for EC2: 0.57838
Training lgb models
Training model for EC1
######################### Training FOLD 1 ########################
auc: 0.70312
######################### Training FOLD 2 ########################
auc: 0.70993
######################### Training FOLD 3 ########################
auc: 0.69467
######################### Training FOLD 4 ########################
auc: 0.69890
######################### Training FOLD 5 ########################
auc: 0.71199
*****************************************
Mean AUC for EC1: 0.70372
Training model for EC2
######################### Training FOLD 1 ########################
auc: 0.56526
######################### Training FOLD 2 ########################
auc: 0.58819
######################### Training FOLD 3 ########################
auc: 0.58677
######################### Training FOLD 4 ########################
auc: 0.58016
######################### Training FOLD 5 ########################
auc: 0.57556
*****************************************
Mean AUC for EC2: 0.57919
Training cat models
Training model for EC1
######################### Training FOLD 1 ########################
auc: 0.70292
######################### Training FOLD 2 ########################
auc: 0.71023
```

```
######################### Training FOLD 3 ########################
auc: 0.69737
######################### Training FOLD 4 ########################
auc: 0.69515
######################### Training FOLD 5 ########################
auc: 0.70850
*****************************************
Mean AUC for EC1: 0.70284
Training model for EC2
######################### Training FOLD 1 ########################
auc: 0.56653
######################### Training FOLD 2 ########################
auc: 0.59321
######################### Training FOLD 3 ########################
auc: 0.60662
######################### Training FOLD 4 ########################
auc: 0.58399
######################### Training FOLD 5 ########################
auc: 0.58688
*****************************************
Mean AUC for EC2: 0.58745
```

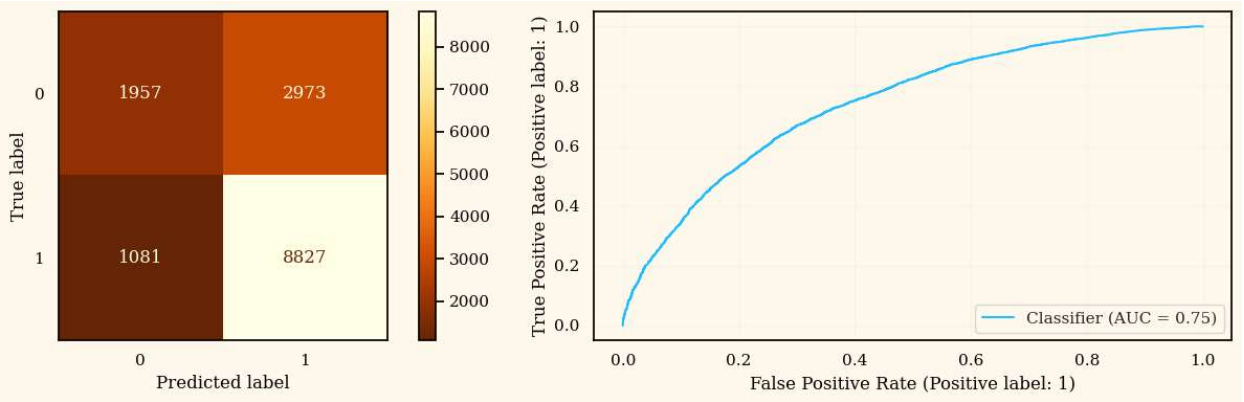
Evaluation and Conclusion

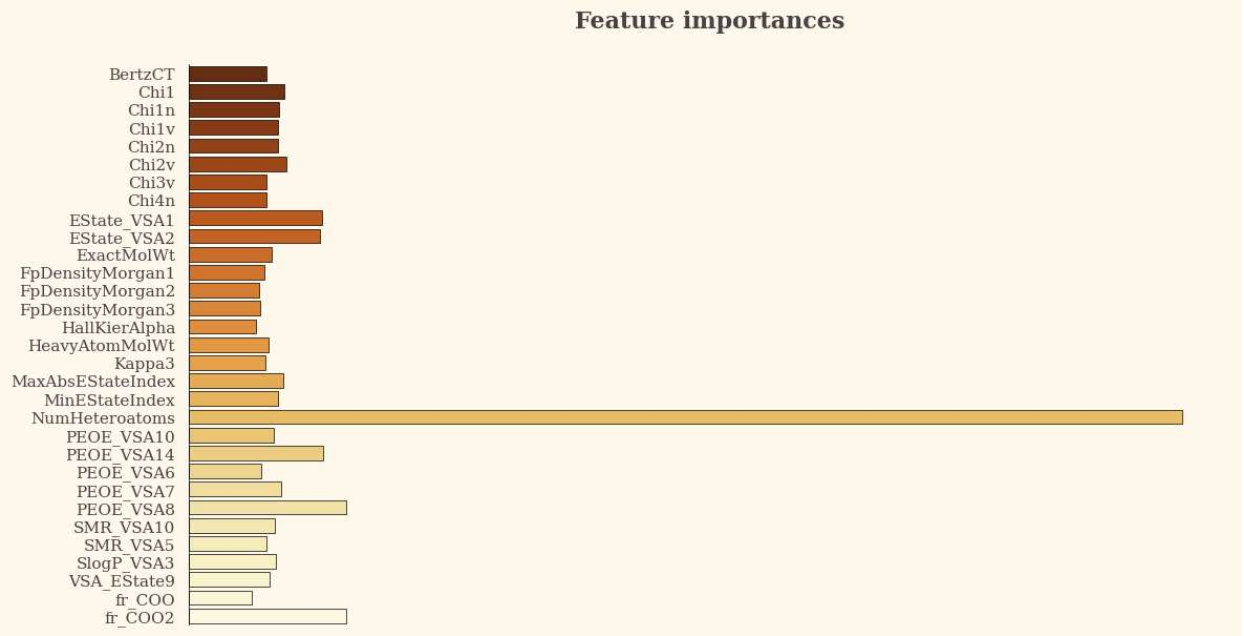
In [13]:
```
# Create a DataFrame for average AUC scores
auc_df = pd.DataFrame(auc_scores)

# Calculate mean AUC for each model and target
auc_df = auc_df.applymap(np.mean)

# Format the numbers to 2 decimal places and apply a gradient color map
styled_auc_df = auc_df.style.format("{:.4f}").background_gradient(cmap='YlOrBr_r')

# Display the styled DataFrame
display(styled_auc_df)
```

	xgb	lgb	cat
EC1	0.7023	0.7037	0.7028
EC2	0.5784	0.5792	0.5874

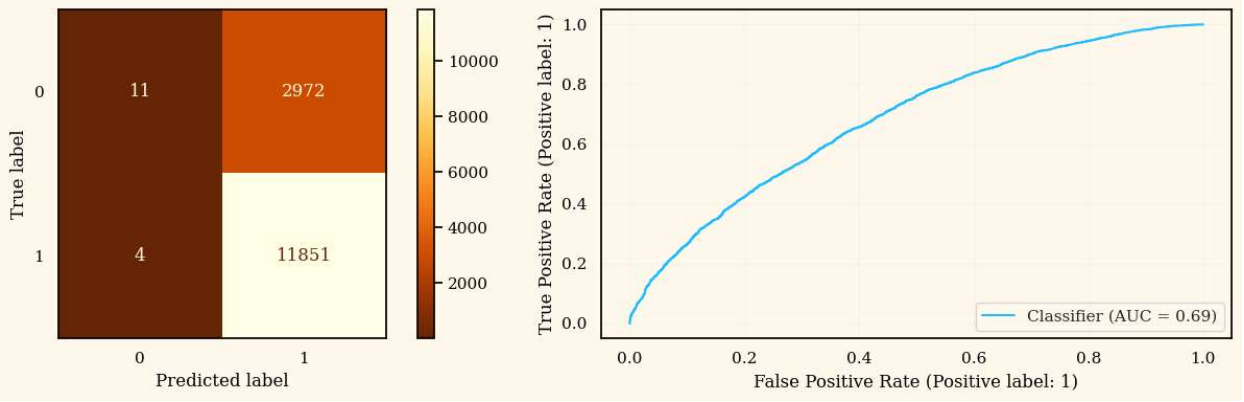
In [14]:
```
for model_name, model_dict in best_models.items():
    for target in target_cols:
        print(f'\033[1;34;4mVisualization for {model_name} {target}\033[0m')  # Blue w
        best_val_preds = model_dict[target].predict_proba(train[features])[:, 1]
        show_confusion_roc(best_val_preds, train[target])
        f_imp_df = pd.DataFrame({'feature': features, 'avg_imp': model_dict[target].fe
        f_importance_plot(f_imp_df)
```

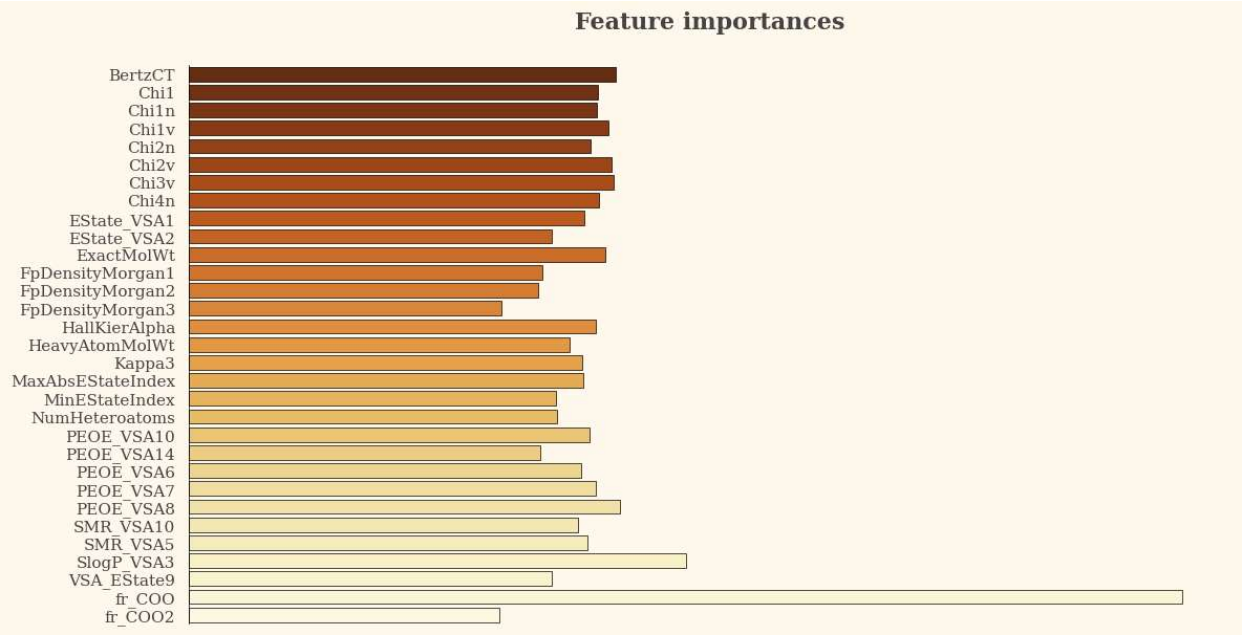
Visualization for xgb EC1



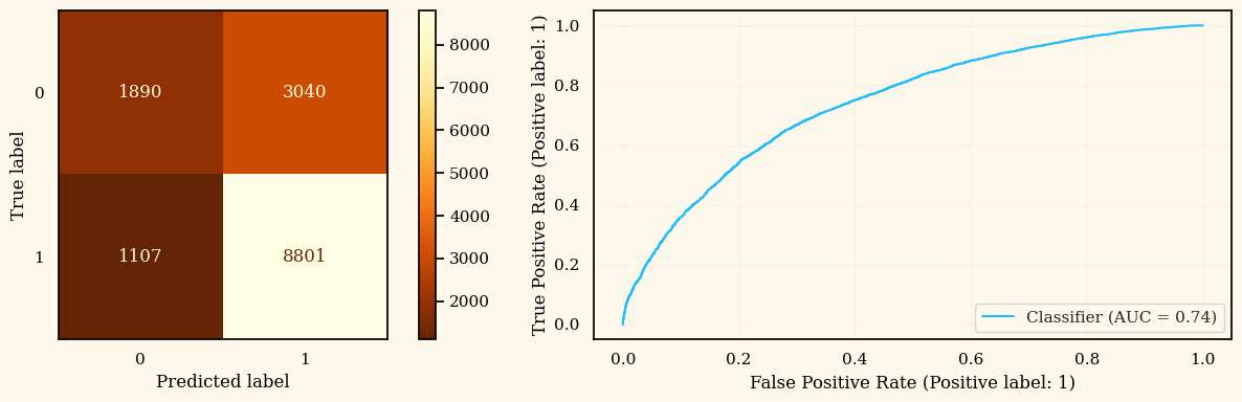


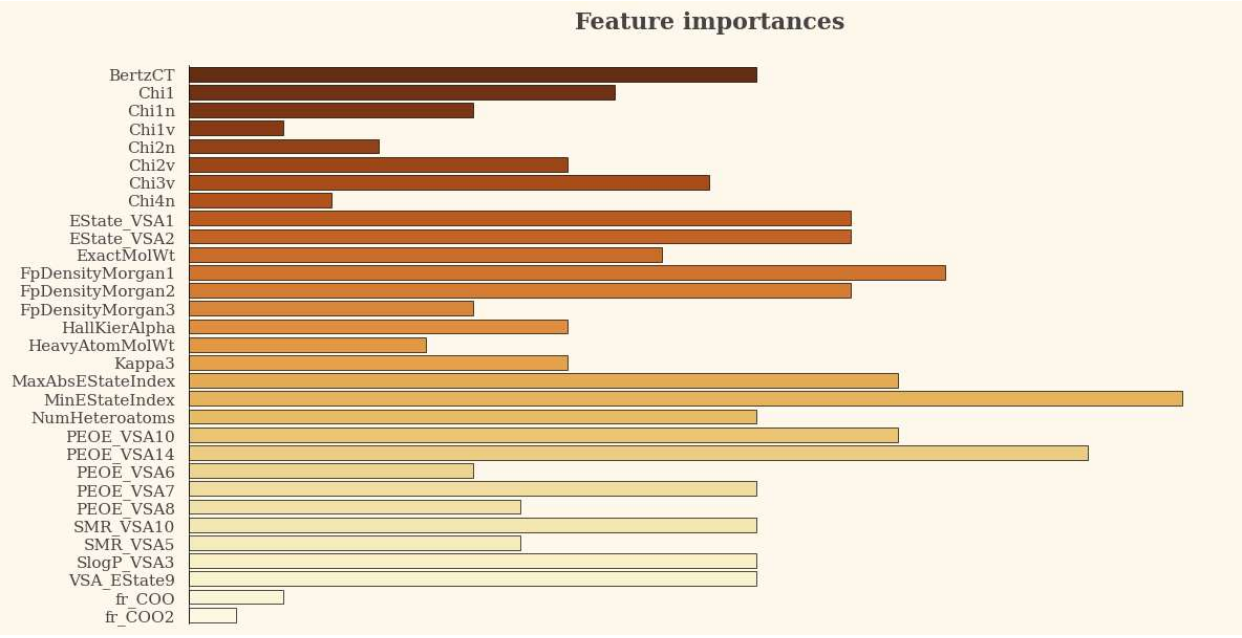
Visualization for xgb EC2



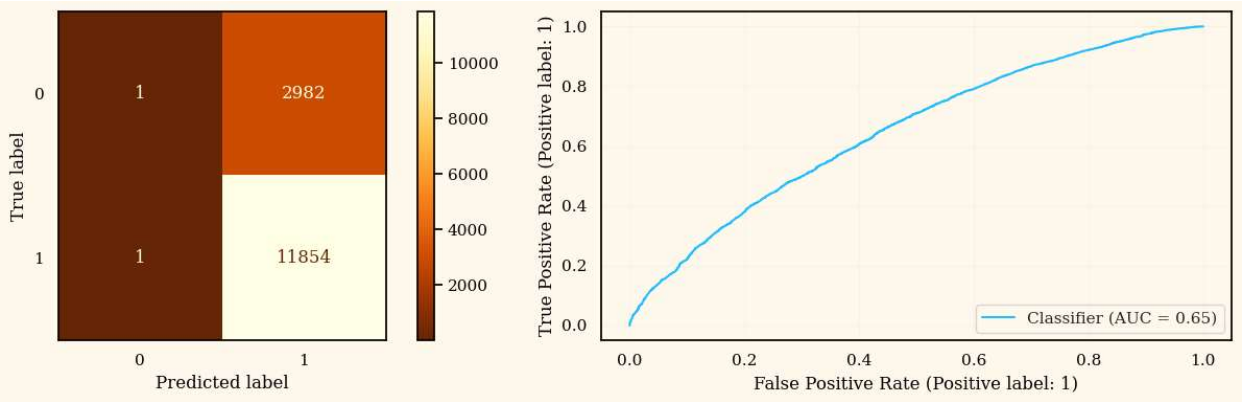


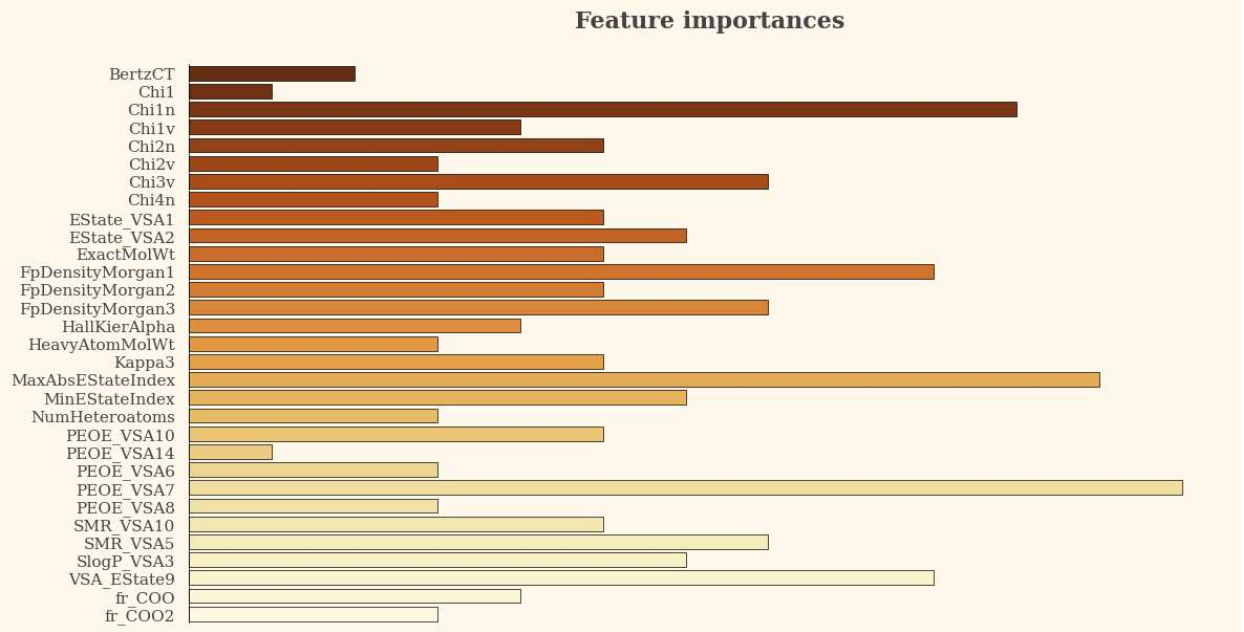
[Visualization for lgb EC1]



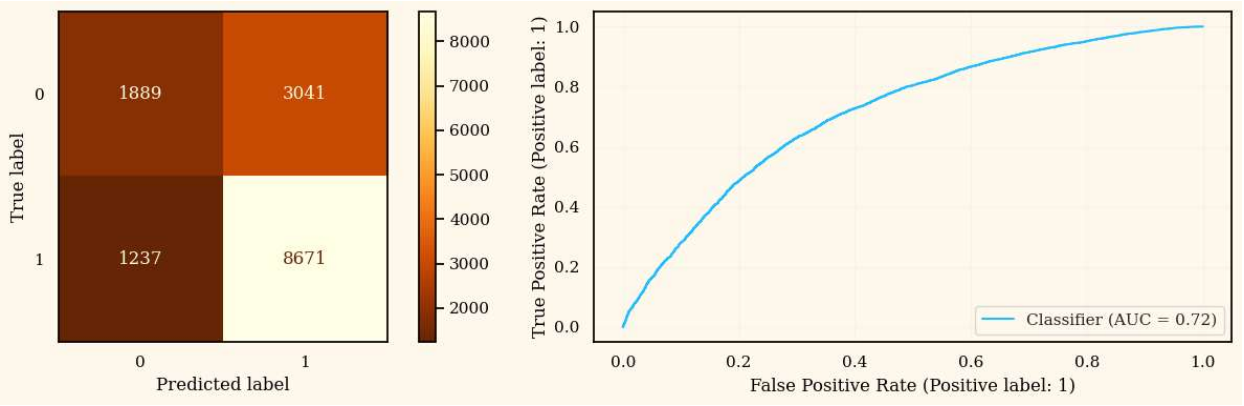


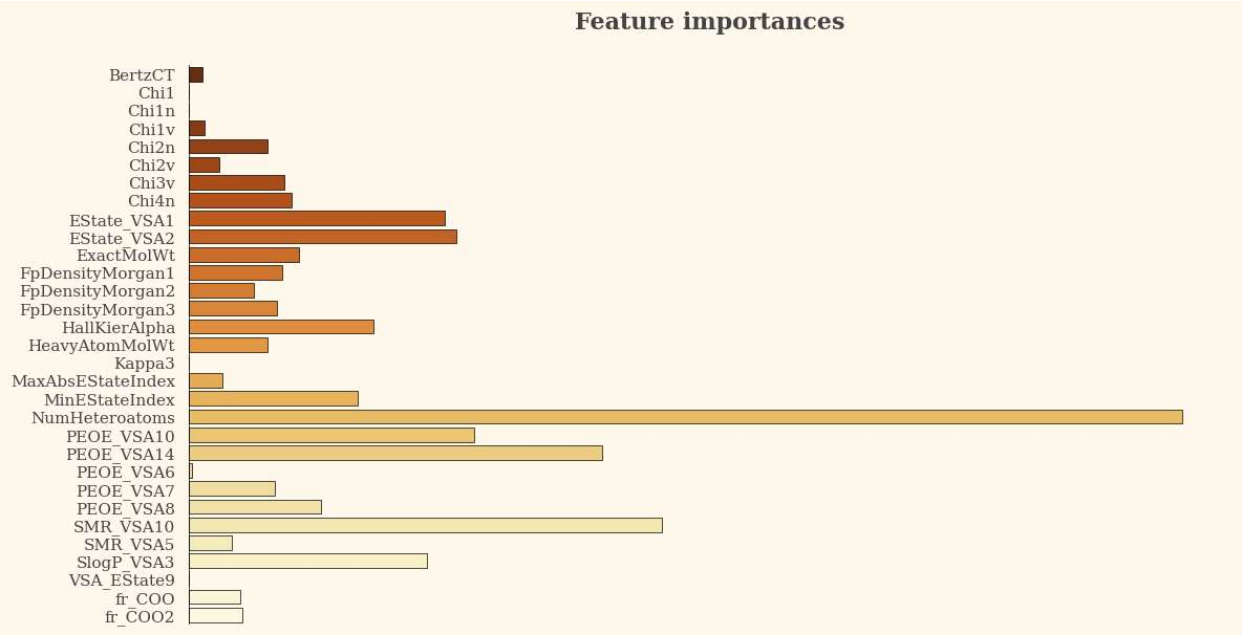
[Visualization for lgb EC2]



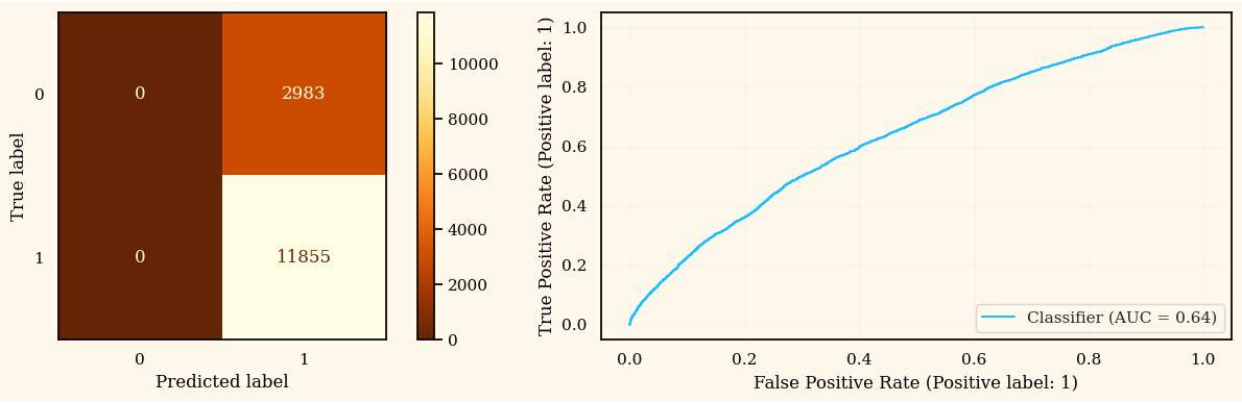


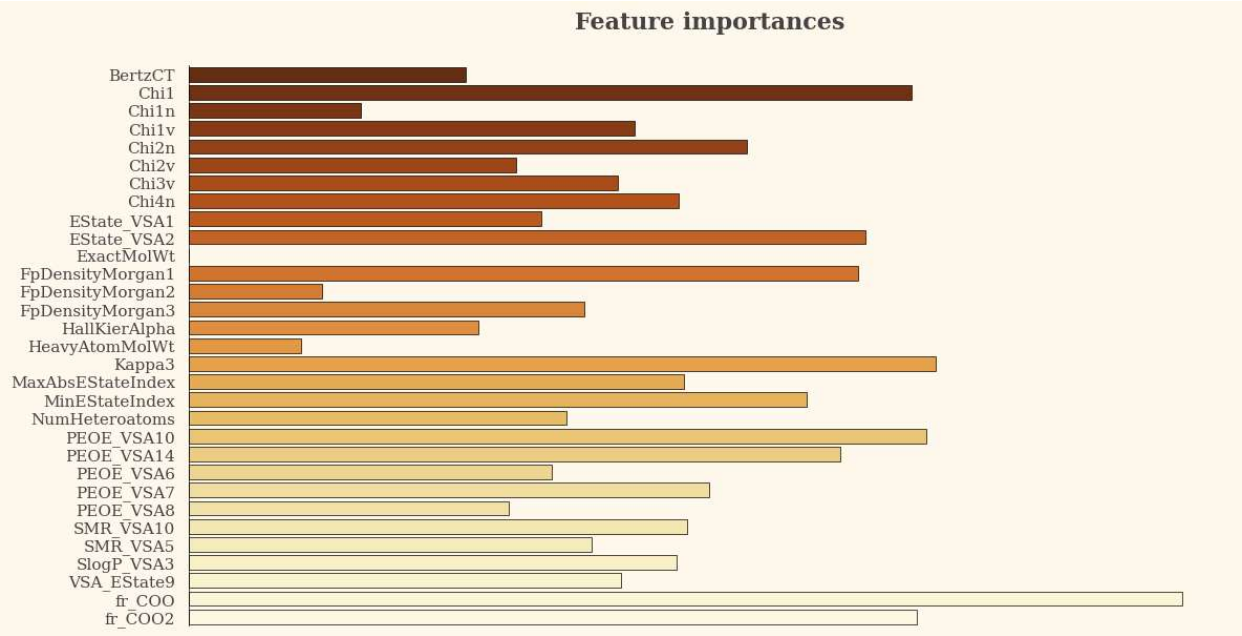
Visualization for cat EC1





[Visualization for cat EC2]





The EC1 model outperforms the EC2 model, despite the fact that the class distribution between EC1 and EC2 is not significantly different, and EC2 has a larger number of instances. This suggests that the difference in performance is likely due to the influence of the features.

Therefore, I encourage you to explore feature engineering techniques to enhance the models' performance.

Furthermore, it's important to note that feature importance varies among the models. This indicates that considering a stacking model, such as an ensemble that combines the predictions of multiple models, may be a valuable avenue for exploration. This approach can potentially further improve predictive performance by leveraging the diverse feature importance patterns of the individual models.