```python
import pandas as pd
import numpy as np
import math
import os
from datetime import datetime
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFECV
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
import tensorflow as tf
import seaborn as sns
directory = 'Forecast_2023'
forecast_files = [file for file in os.listdir(directory) if 'Forecast' in file]
merged_forecast = pd.DataFrame()
for file in forecast_files:
file_path = os.path.join(directory, file)
forecast_data = pd.read_csv(file_path)
merged_forecast = pd.concat([merged_forecast, forecast_data], ignore_index=True)
merged_forecast['time'] = pd.to_datetime(merged_forecast['time'])
merged_forecast = merged_forecast.sort_values('time')
merged_forecast = merged_forecast.drop(columns=['latitude', 'longitude'])
merged_forecast['t2m'] = merged_forecast['t2m'] - 273.15
merged_forecast
vlinder_data = pd.read_csv('vlinder19_2023.csv')
vlinder_data['datetime'] = pd.to_datetime(vlinder_data['datetime'])
merged_data = pd.merge(merged_forecast, vlinder_data, left_on='time', right_on='datetime',
how='inner')
merged_data.drop('datetime', axis=1, inplace=True)
merged_data = merged_data.dropna()
print(merged_data['t2m'].head())
print(merged_data['temp'].head())
merged_datatotal_length = len(merged_data)
train_size = int(total_length * 0.6)
validate_size = int(total_length * 0.2)
test_size = total_length - train_size - validate_size
trainset = merged_data[:train_size]
validateset = merged_data[train_size:train_size + validate_size]
testset = merged_data[train_size + validate_size:]
print(f"Training set size: {len(trainset)}")
print(f"Validation set size: {len(validateset)}")
print(f"Test set size: {len(testset)}")
feature_columns = ['cape', 'sp', 'tcw', 'sshf', 'slhf', 'msl', 'u10', 'v10', 't2m',
'd2m', 'tp', 'sf', 'ttr', 'str', 'ssr',
'skt', 'cin', 'sm', 'st', 'sd', 'tcc', 'tp', 'mx2t6', 'mn2t6']
all_data = merged_data[feature_columns + ['temp']]
correlation_matrix = all_data.corr()
correlation_with_temp = correlation_matrix['temp'].drop('temp').sort_values(ascending=False)
plt.figure(figsize=(10, 8))
correlation_with_temp.plot(kind='bar')
plt.title('Correlation of Features with Temperature (temp)')
plt.ylabel('Correlation coefficient')
plt.xlabel('Features')
plt.show()
plt.figure(figsize=(14, 12))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True,
square=True)
plt.title('Heat Map of Correlation Matrix')
plt.show()
train_features = trainset[feature_columns]
train_target = trainset['temp']
validate_features = validateset[feature_columns]
validate_target = validateset['temp']
linear_regression_model = LinearRegression()
linear_regression_model.fit(train_features, train_target)
feature_coefficients = linear_regression_model.coef_
importance_df = pd.DataFrame({'Feature': feature_columns,
'Coefficient': feature_coefficients
})
importance_df['Absolute Coefficient'] = importance_df['Coefficient'].abs()
importance_df = importance_df.sort_values(by='Absolute Coefficient', ascending=False)
top_10_features = importance_df.head(10)
top_10_feature_list = top_10_features['Feature'].tolist()
plt.figure(figsize=(10, 8))
plt.bar(importance_df['Feature'], importance_df['Coefficient'])
plt.xticks(rotation=90)
plt.title('Top 10 Feature Importance with Linear Regression')
plt.ylabel('Coefficient')
plt.xlabel('Features')
plt.show()
print("Top 10 Features:")
print(top_10_feature_list)
predictions = linear_regression_model.predict(validate_features)
mse = mean_squared_error(validate_target, predictions)
print(f"Mean Squared Error on validation set using all features: {mse}")
linear_regression_model_2 = LinearRegression()
rfecv_lr = RFECV(estimator=linear_regression_model_2, step=1, cv=10,
scoring='neg_mean_squared_error')
rfecv_lr.fit(train_features, train_target)
selected_features = np.array(feature_columns)[rfecv_lr.support_].tolist()
print(f"Selected features: {selected_features}")
```

```python
predictions = rfecv_lr.predict(validate_features)
mse = mean_squared_error(validate_target, predictions)
print(f"Mean Squared Error on validation set using RFECV: {mse}")print("Top 10 features farthest from zero for evaluation with linear regression:",
top_10_feature_list)
selected_train_features = trainset[top_10_feature_list]
selected_validate_features = validateset[top_10_feature_list]
optimized_linear_regression_model = LinearRegression()
optimized_linear_regression_model.fit(selected_train_features, train_target)
validate_predictions = optimized_linear_regression_model.predict(selected_validate_features)
mse = mean_squared_error(validate_target, validate_predictions)
print(f"Validation MSE: {mse}")
top_10_features = correlation_with_temp.abs().nlargest(10).index.tolist()
print("Top 10 features farthest from zero for evaluation:", top_10_features)
selected_train_features = trainset[top_10_features]
selected_validate_features = validateset[top_10_features]
second_optimized_linear_regression_model = LinearRegression()
second_optimized_linear_regression_model.fit(selected_train_features, train_target)
validate_predictions =
second_optimized_linear_regression_model.predict(selected_validate_features)
mse = mean_squared_error(validate_target, validate_predictions)
print(f"Validation MSE: {mse}")
first_rf_model = RandomForestRegressor(random_state=42)
first_rf_model.fit(train_features, train_target)
feature_importances = first_rf_model.feature_importances_
importance_df = pd.DataFrame({
'Feature': feature_columns,
'Importance': feature_importances
})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 8))
plt.bar(importance_df['Feature'], importance_df['Importance'])
plt.xticks(rotation=90)
plt.title('Feature Importance with Random Forest')
plt.ylabel('Importance')
plt.xlabel('Features')
plt.show()first_rf_model = RandomForestRegressor(random_state=42)
first_rf_model.fit(train_features, train_target)
val_predictions = first_rf_model.predict(validate_features)
mse = mean_squared_error(validate_target, val_predictions)
print(f'Mean Squared Error: {mse}')
top_10_feature_names = correlation_with_temp.abs().nlargest(10).index.tolist()
print("Top 10 features farthest from zero for evaluation:", top_10_feature_names)
top_10_train_features = train_features[top_10_feature_names]
top_10_validate_features = validate_features[top_10_feature_names]
second_rf_model = RandomForestRegressor(random_state=42)
param_grid = {
'n_estimators': [50, 100, 200],
'max_depth': [10, 20, 30, None],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'bootstrap': [True, False]
}
grid_search = GridSearchCV(estimator=second_rf_model, param_grid=param_grid, cv=5,
n_jobs=-1, scoring='neg_mean_squared_error')
grid_search.fit(top_10_train_features, train_target)
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")
best_second_rf_model = RandomForestRegressor(**best_params, random_state=42)
best_second_rf_model.fit(top_10_train_features, train_target)
validate_predictions1 = best_second_rf_model.predict(top_10_validate_features)
mse = mean_squared_error(validate_target, validate_predictions1)
print(f"Mean Squared Error on validation set: {mse}")fourth_rf_model = RandomForestRegressor(random_state=42)
param_grid = {
'n_estimators': [50, 100, 200],
'max_depth': [10, 20, 30, None],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'bootstrap': [True, False]
}
grid_search = GridSearchCV(estimator=fourth_rf_model, param_grid=param_grid, cv=5,
n_jobs=-1, scoring='neg_mean_squared_error')
grid_search.fit(train_features, train_target)
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")
best_fourth_rf_model = RandomForestRegressor(**best_params, random_state=42)
best_fourth_rf_model.fit(train_features, train_target)
validate_predictions = best_fourth_rf_model.predict(validate_features)
mse = mean_squared_error(validate_target, validate_predictions)
print(f"Mean Squared Error on validation set: {mse}")
important_features = ['st', 't2m']
important_train_features = train_features[important_features]
important_validate_features = validate_features[important_features]
rf_model = RandomForestRegressor(random_state=42)
param_grid = {
'n_estimators': [50, 100, 200],
'max_depth': [10, 20, 30, None],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'bootstrap': [True, False]
}
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1,
scoring='neg_mean_squared_error')
grid_search.fit(important_train_features, train_target)
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")
best_rf_model = RandomForestRegressor(**best_params, random_state=42)
```

```
best_rf_model.fit(important_train_features, train_target)
validate_predictions = best_rf_model.predict(important_validate_features)
mse = mean_squared_error(validate_target, validate_predictions)
print(f"Mean Squared Error on validation set: {mse}")np.random.seed(42)
tf.random.set_seed(42)
top_10_feature_names = correlation_with_temp.abs().nlargest(10).index.tolist()
print("Top 10 features farthest from zero for evaluation:", top_10_feature_names)
top_10_train_features = train_features[top_10_feature_names]
top_10_validate_features = validate_features[top_10_feature_names]
scaler = StandardScaler()
train_features_scaled = scaler.fit_transform(top_10_train_features)
validate_features_scaled = scaler.transform(top_10_validate_features)
model = Sequential([
Dense(64, activation='relu', input_shape=(train_features_scaled.shape[1],)),
Dense(32, activation='relu'),
Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(train_features_scaled, train_target, epochs=50, batch_size=32, verbose=1,
validation_split=0.1)
mse = model.evaluate(validate_features_scaled, validate_target, verbose=0)
print(f"Mean Squared Error on Validation Set: {mse}")
np.random.seed(42)
tf.random.set_seed(42)
scaler = StandardScaler()
train_features_scaled = scaler.fit_transform(train_features)
validate_features_scaled = scaler.transform(validate_features)
model_two = Sequential([
Dense(64, activation='relu', input_shape=(train_features.shape[1],)),
Dense(32, activation='relu'),
Dense(1)
])
model_two.compile(optimizer='adam', loss='mean_squared_error')
model_two.fit(train_features_scaled, train_target, epochs=50, batch_size=32, verbose=1,
validation_split=0.1)
mse = model_two.evaluate(validate_features_scaled, validate_target, verbose=0)
print(f"Mean Squared Error on Validation Set: {mse}")def create_model(neurons=32, optimizer='adam'):
model = Sequential([
Dense(neurons, activation='relu', input_shape=(train_features_scaled.shape[1],)),
Dense(neurons, activation='relu'),
Dense(1)
])
model.compile(optimizer=optimizer, loss='mean_squared_error')
return model
model = KerasRegressor(build_fn=create_model, verbose=0)
param_grid = {
'neurons': [32, 64, 128],
'batch_size': [16, 32, 64],
'epochs': [50, 100],
'optimizer': ['adam', 'rmsprop']
}
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
n_iter=10, cv=3, verbose=2, n_jobs=-1,scoring='neg_mean_squared_error')
random_search_result = random_search.fit(train_features_scaled, train_target)
print(f"Best: {random_search_result.best_score_} using
{random_search_result.best_params_}")
selected_test_features = testset[top_10_features]
test_target = testset['temp']
test_predictions = second_optimized_linear_regression_model.predict(selected_test_features)
test_mse = mean_squared_error(test_target, test_predictions)
print(f"Test MSE: {test_mse}")
plt.figure(figsize=(10, 6))
plt.scatter(test_target, test_predictions, alpha=0.5, color='b')
plt.plot([min(test_target), max(test_target)], [min(test_target), max(test_target)], 'r--', lw=2)
plt.xlabel('Actual Temperature')
plt.ylabel('Predicted Temperature')
plt.title('Actual vs Predicted Temperature')
plt.grid(True)
plt.show()df = pd.read_csv("Forecast_27_04_2024.csv")
df['time'] = pd.to_datetime(df['time'])
df = df[df['time'].dt.time != pd.to_datetime('00:00:00').time()]
features = top_10_features
new_forecast = df[features]
predicted_temps = second_optimized_linear_regression_model.predict(new_forecast)
df['predicted_temp'] = predicted_temps
print(df.head())
print(merged_data['t2m'].head())
print(merged_data['temp'].head())
forecast_init_time = datetime.strptime('2024-04-27', '%Y-%m-%d')
result_csv = pd.DataFrame()
for time in df['time'].unique():
time = pd.to_datetime(time)
df_time = df[df['time'] == time]
horizon_hours = (time - forecast_init_time).total_seconds() / 3600
corrected_temps = df_time['predicted_temp'].values
quantiles = [np.quantile(corrected_temps, q) for q in [0.025, 0.25, 0.5, 0.75, 0.975]]
new_row = {
'forecast_date': time.strftime('%Y-%m-%d %H:%M:%S'),
'target': 't2m',
'horizon': f"{int(horizon_hours)} hour",
'q0.025': quantiles[0],
'q0.25': quantiles[1],
'q0.5': quantiles[2],
'q0.75': quantiles[3],
'q0.975': quantiles[4]
}
```

```
result_csv = pd.concat([result_csv, pd.DataFrame([new_row])], ignore_index=True)
result_csv.to_csv('20240427_balmorhea.csv', index=False)
result_csv
```