Secondo Modello

In questo notebook sono usati i dataset su cui è stata operata la fase preliminare di **preparazione dei dati**, la **divisione** in base alla regione e la **selezione delle feature**.

È costruita una **foresta**.

È impiegato un algoritmo per fare **tuning dell'iperparametro `n_estimators`** ossia il numero di alberi della foresta. Per fare tuning dei parametri si fa uso di un **dataset di validation**: si mantiene l'iperparametro che produce il minimo errore su validation.

In [1]:
```
# Librerie
import pandas            as pd
import numpy             as np
import matplotlib.pyplot as plt
import warnings

from sklearn.metrics     import mean_squared_error
from sklearn.ensemble    import RandomForestRegressor
from sklearn.utils       import resample

warnings.filterwarnings('ignore')
```

Lettura dei dati

In [2]:
```
# local file path

dir_name     = 'selezione'
region_names = np.array(['A', 'B', 'C'])
region_ids   = np.array(['1286', '2061', '3101'])

fp_Xtrain = []
fp_Xval   = []
fp_Xtest  = []
fp_ytrain = []
fp_yval   = []
fp_ytest  = []

for i in range(3):
    fp_Xtrain.append(dir_name + f'/X_train{region_names[i]}.csv')
    fp_Xval  .append(dir_name + f'/X_val{  region_names[i]}.csv')
    fp_Xtest .append(dir_name + f'/X_test{ region_names[i]}.csv')
    fp_ytrain.append(dir_name + f'/y_train{region_names[i]}.csv')
    fp_yval  .append(dir_name + f'/y_val{  region_names[i]}.csv')
    fp_ytest .append(dir_name + f'/y_test{ region_names[i]}.csv')
```

Lettura dei dataset su cui è stata fatta la selezione delle feature.

In [3]:
```
# Lettura dei dati

X_train = []
X_val   = []
X_test  = []
y_train = []
y_val   = []
y_test  = []

for i in range(3):
    X_train.append(pd.read_csv(fp_Xtrain[i], low_memory=False))
    X_val  .append(pd.read_csv(fp_Xval  [i], low_memory=False))
    X_test .append(pd.read_csv(fp_Xtest [i], low_memory=False))
    y_train.append(pd.read_csv(fp_ytrain[i], low_memory=False))
    y_val  .append(pd.read_csv(fp_yval  [i], low_memory=False))
    y_test .append(pd.read_csv(fp_ytest [i], low_memory=False))

X_train = np.array(X_train, dtype=object)
X_val   = np.array(X_val,   dtype=object)
X_test  = np.array(X_test,  dtype=object)
y_train = np.array(y_train, dtype=object)
y_val   = np.array(y_val,   dtype=object)
y_test  = np.array(y_test,  dtype=object)
```

In [4]:
```
def dimensionality(y=False):
    for i in range(3):
        print(f'X_train{region_names[i]}: {X_train[i].shape}')
        print(f'X_val{region_names[i]}:   {X_val  [i].shape}')
        print(f'X_test{region_names[i]}:  {X_test [i].shape}')
        if y:
            print(f'y_train{region_names[i]}: {y_train[i].shape}')
            print(f'y_val{region_names[i]}:   {y_val  [i].shape}')
            print(f'y_test{region_names[i]}:  {y_test [i].shape}')
        print()
```

In [5]:
```
dimensionality(y=True)
```
```
X_trainA: (26819, 55)
X_valA:   (9006, 55)
X_testA:  (9085, 55)
y_trainA: (26819, 1)
y_valA:   (9006, 1)
y_testA:  (9085, 1)

X_trainB: (8119, 32)
X_valB:   (2658, 32)
X_testB:  (2606, 32)
y_trainB: (8119, 1)
y_valB:   (2658, 1)
y_testB:  (2606, 1)

X_trainC: (64771, 33)
X_valC:   (21908, 33)
X_testC:  (21876, 33)
y_trainC: (64771, 1)
y_valC:   (21908, 1)
y_testC:  (21876, 1)
```

Variabili Globali

In [6]:
```
# Nomi delle regioni
region_ids = np.array(['1286', '2061', '3101'])

# Riduzione del dataset C
C_PERC = 2/5
C_IND  = 2

# Percentuale per il sottoinsieme su cui costruire il modello
#SUB_PERC = [1/1000, 1/100, 1/1000] # A:   100, B:  100, C:   100; Test rapido      ~  5 min
#SUB_PERC = [   1/3,     1,    1/3] # A:  8000, B: 8000, C:  8000; Test medio       ~  1 ora
SUB_PERC = [     1,     1,      1] # A: 24000, B: 8000, C: 64000; Dataset Completo ~  5 ore

# Random Forest Regressor
RF_START = 50
RF_END   = 1000
RF_STEP  = 50
```

Limite del numero di istanze del terzo dataset

Per limiti tecnici di memoria, la gestione del dataset C risulta complicata in quato incorre spesso in errori a run-time di cateoria **MemoryError**: la macchina su cui è eseguito il notebook non è in grado di allocare la quantità di memoria richiesta per lavorare sull'intero dataset di Train (circa 64000 righe per 33 colonne).

È dunque inevitabile dover ridurre l'insieme con un numero di righe il cui calcolo è supportato, cioè circa il 40% del dataset. Il dataset per la terza regione (`regionidcounty` : 3101) è riscalato su 2/5 delle osservazioni per Train, Validation e Test.

L'operazione introduce dunque una forte approssimazione per questa regione, che per limiti tecnici è però inevitabile.

In [7]:
```
for X, y in zip([X_train, X_val, X_test], [y_train, y_val, y_test]):
    X[C_IND], y[C_IND] = resample(
        X[C_IND], y[C_IND],
        n_samples = int(C_PERC * len(X[2]))
    )
```

In [8]:
```
dimensionality(y=True)
```
```
X_trainA: (26819, 55)
X_valA:   (9006, 55)
X_testA:  (9085, 55)
y_trainA: (26819, 1)
y_valA:   (9006, 1)
y_testA:  (9085, 1)

X_trainB: (8119, 32)
X_valB:   (2658, 32)
X_testB:  (2606, 32)
y_trainB: (8119, 1)
y_valB:   (2658, 1)
y_testB:  (2606, 1)

X_trainC: (25908, 33)
X_valC:   (8763, 33)
X_testC:  (8750, 33)
y_trainC: (25908, 1)
y_valC:   (8763, 1)
y_testC:  (8750, 1)
```

Definizione di un Subset per la costruzione del modello

Gli algoritmi usati in questo notebook hanno un costo computazionale elevato, per questo è definito un sottoinsieme dei dataset originali su cui far girare gli algoritmi; questo per facilitare la fase di creazione e di testing e ottenere risultati verosimili in tempi utili a verificare il corretto funzionamento del processo.

Nella versione finale gli algoritmi useranno la totalità delle istanze del dataset originale.

In [9]:
```
for i in range(3):
    print(f'Region {region_names[i]}')
    print(int(len(X_train[i])*SUB_PERC[i]))
    print(int(len(X_val  [i])*SUB_PERC[i]))
    print()
```
```
Region A
26819
9006

Region B
8119
2658

Region C
25908
8763
```

In [10]:
```
X_train_sub = []
y_train_sub = []
X_val_sub   = []
y_val_sub   = []
```

In [11]:
```
for i in range(3):
    Xt_sub, yt_sub = resample(X_train[i], y_train[i], n_samples = int(SUB_PERC[i]*len(X_train[i])))
    Xv_sub, yv_sub = resample(X_val  [i], y_val  [i], n_samples = int(SUB_PERC[i]*len(X_val[i])))
    X_train_sub.append(Xt_sub)
    y_train_sub.append(yt_sub)
    X_val_sub  .append(Xv_sub)
    y_val_sub  .append(yv_sub)
```

In [12]:
```
def dimensionality_sub(y=False):
    for i in range(3):
        print(f'X_train_sub{region_names[i]}: {X_train_sub[i].shape}')
        print(f'X_val_sub{region_names[i]}:   {X_val_sub  [i].shape}')
        if y:
            print(f'y_train_sub{region_names[i]}: {y_train_sub[i].shape}')
            print(f'y_val_sub{region_names[i]}:   {y_val_sub  [i].shape}')
        print()
```

In [13]:
```
dimensionality_sub(y=True)
```
```
X_train_subA: (26819, 55)
X_val_subA:   (9006, 55)
y_train_subA: (26819, 1)
y_val_subA:   (9006, 1)

X_train_subB: (8119, 32)
X_val_subB:   (2658, 32)
y_train_subB: (8119, 1)
y_val_subB:   (2658, 1)

X_train_subC: (25908, 33)
X_val_subC:   (8763, 33)
y_train_subC: (25908, 1)
y_val_subC:   (8763, 1)
```

Costruzione Random Forest Regressor

Tuning numero di alberi attraverso il parametro `n_estimators`. Per studiare l'andamento la funzione produce due grafici che illustrano l'andamento di $bias^2$, *varianza* e *mean squared error* in funzione del numero di alberi sia per l'insieme di *Train* che per quello di *Validation*.

In [14]:
```
plt.rcParams.update({'font.size': 35})
```

In [15]:
```
def get_bias_var_mse(X, y, model):
        y_pred = model.predict(X)
        return {
            'bias': ((y - np.mean(y_pred))**2).mean(),\
            'var':  np.var(y_pred).mean(),\
            'mse':  ((y_pred - y.reshape(-1,1))**2).mean()
        }
```

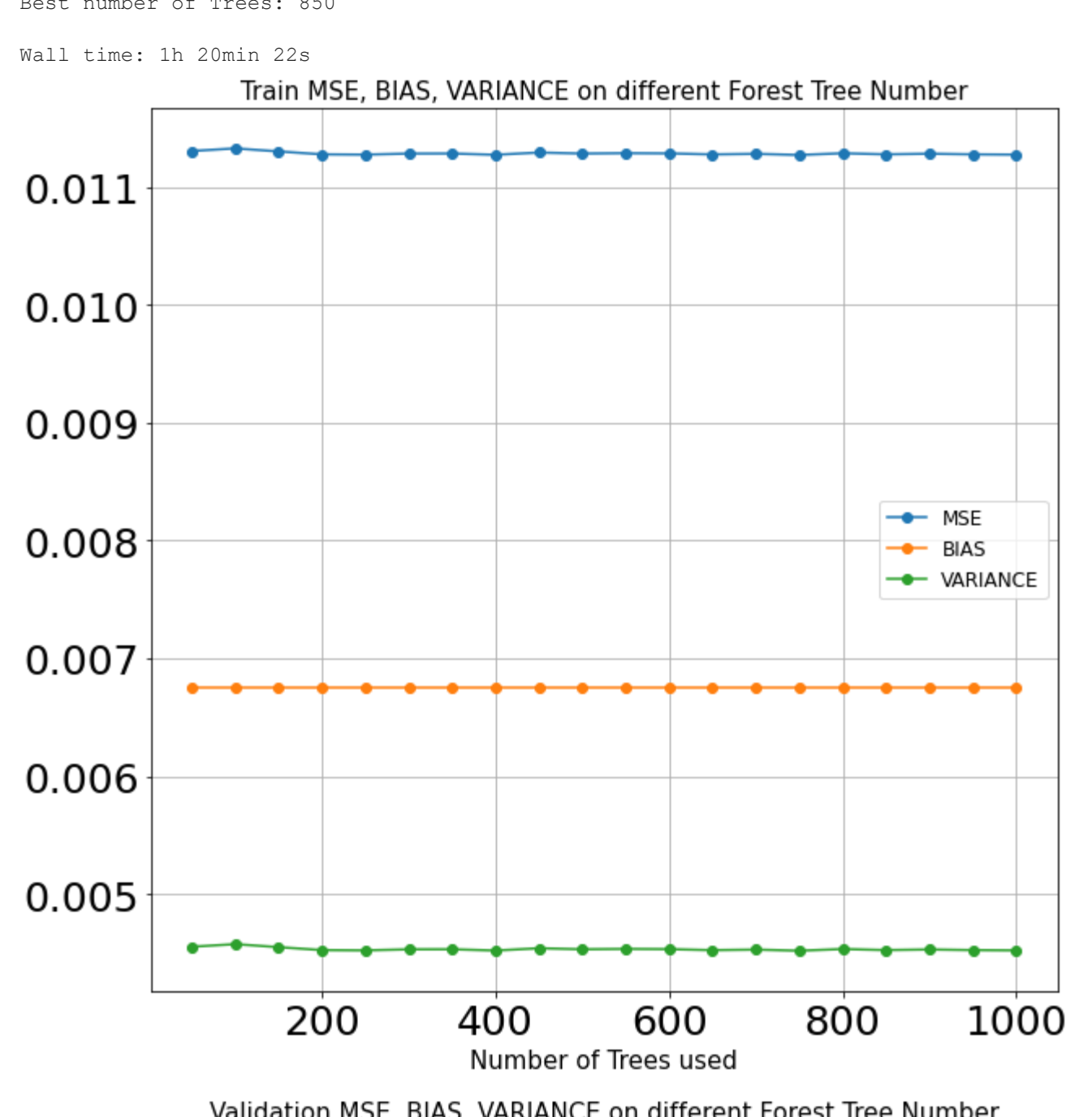
In [16]:
```
# Costruzione RandomForestRegressor
def RandomForestRegressor_validation(X_train, y_train, X_val, y_val, verbose=False, debug=False, file_name = ''):

    def get_rf_reg(estimator):
        dt = RandomForestRegressor(
            n_estimators=estimator,
            # criterion = 'squared_error',
            n_jobs=-1
        )
        dt.fit(X_train, y_train)
        return dt

    def bias_var_mse(X, y, model):
        stats = get_bias_var_mse(X, y, model)
        return stats['bias'],\
               stats['var'],\
               stats['mse']

    def plot_mse(stats, name):

        print(f'{name}: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA')
        print()

        for n in ['mse', 'bias', 'var']:

            min_  = min(stats[n])
            best  = (np.argmin(stats[n]) * RF_STEP) + RF_START

            print(f'Punteggio finale: {stats[n][-1]} ({RF_END}) alberi')
            print(f'Best {n}: {min_}')
            print(f'Best number of Trees: {best}')
            print()

        fig, ax = plt.subplots(figsize=(len(stats['mse'])/2, 10))

        ax.tick_params(axis='both', which='major', labelsize=25)
        ax.tick_params(axis='both', which='minor', labelsize=15)

        ax.plot(range(RF_START, RF_END+1, RF_STEP),  stats['mse' ], 'o-', label='MSE')
        ax.plot(range(RF_START, RF_END+1, RF_STEP),  stats['bias'], 'o-', label='BIAS')
        ax.plot(range(RF_START, RF_END+1, RF_STEP),  stats['var' ], 'o-', label='VARIANCE')

        ax.set_title(f"{name} MSE, BIAS, VARIANCE on different Forest Tree Number", fontsize=15)
        ax.set_xlabel("Number of Trees used", fontsize=15)
        ax.grid()
        ax.legend(prop={'size': 12})

        if file_name != '':
            fig.savefig('images/' + file_name + '_RandomForestRegressor' + name + '.jpg')

    y_train = y_train.values.ravel()
    y_val   = y_val  .values.ravel()

    first = True

    info = []

    train_stats  = {
        'bias' : [],
        'var'  : [],
        'mse'  : []
    }

    val_stats    = {
        'bias' : [],
        'var'  : [],
        'mse'  : []
    }

    for estimator in range(RF_START, RF_END+1, RF_STEP):

        if debug:
            print(f'{estimator}/{RF_END}')

        model = get_rf_reg(estimator)

        trn_bias, trn_var, trn_mse = bias_var_mse(X_train, y_train, model)
        val_bias, val_var, val_mse = bias_var_mse(  X_val,   y_val, model)

        train_stats['bias'].append(trn_bias)
        train_stats['var' ].append(trn_var )
        train_stats['mse' ].append(trn_mse )

        val_stats['bias'].append(val_bias)
        val_stats['var' ].append(val_var )
        val_stats['mse' ].append(val_mse )

        info.append(f'Estimators: {estimator}' +\
                    f'\n(Train MSE:       {trn_mse }  - Val MSE:      {val_mse })' +\
                    f'\n(Train Bias:      {trn_bias}  - Val Bias:     {val_bias})' +\
                    f'\n(Train Variance:  {trn_var }  - Val Variance: {val_var })' )

        if(first or val_mse < best_mse):
            first = False
            best_mse       = val_mse
            best_estimator = estimator
            best_model     = model

    if verbose:
        print()
        print("MSE, BIAS, VARIANCE Train e Validation")
        print(*info, sep='\n')
        print()

    plot_mse(train_stats, "Train"     )
    plot_mse(  val_stats, "Validation")

    return best_model
```

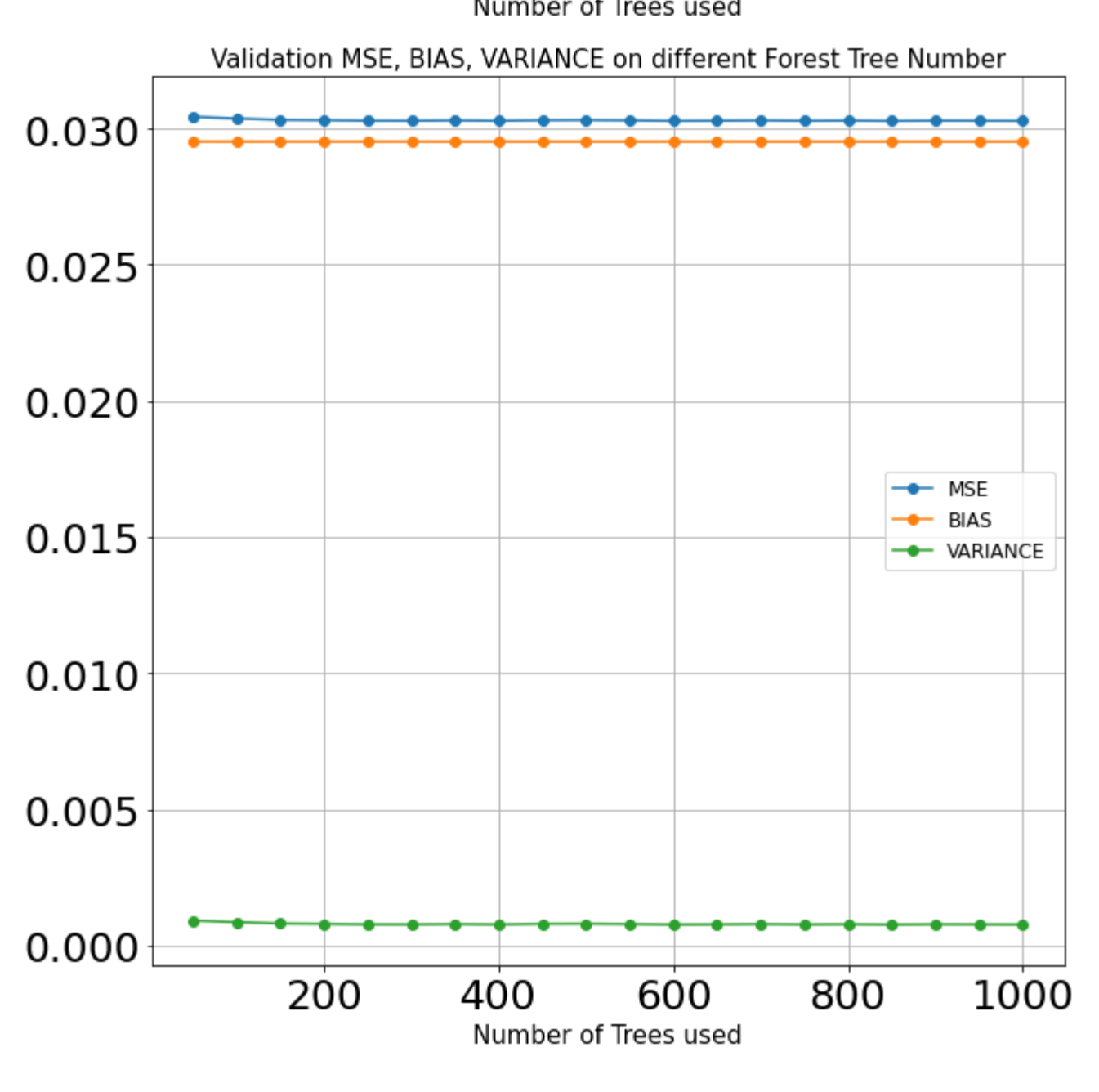
In [17]:
```
def get_rf(index, verbose=False, debug=False, file_name=''):
    if file_name == '':
        file_name = region_ids[index]
    return RandomForestRegressor_validation(
        X_train_sub[index],
        y_train_sub[index],
        X_val_sub  [index],
        y_val_sub  [index],
        verbose    = verbose,
        debug      = debug,
        file_name = file_name
    )
```

In [18]:
```
rf_model = []
```

Prima Contea 1286

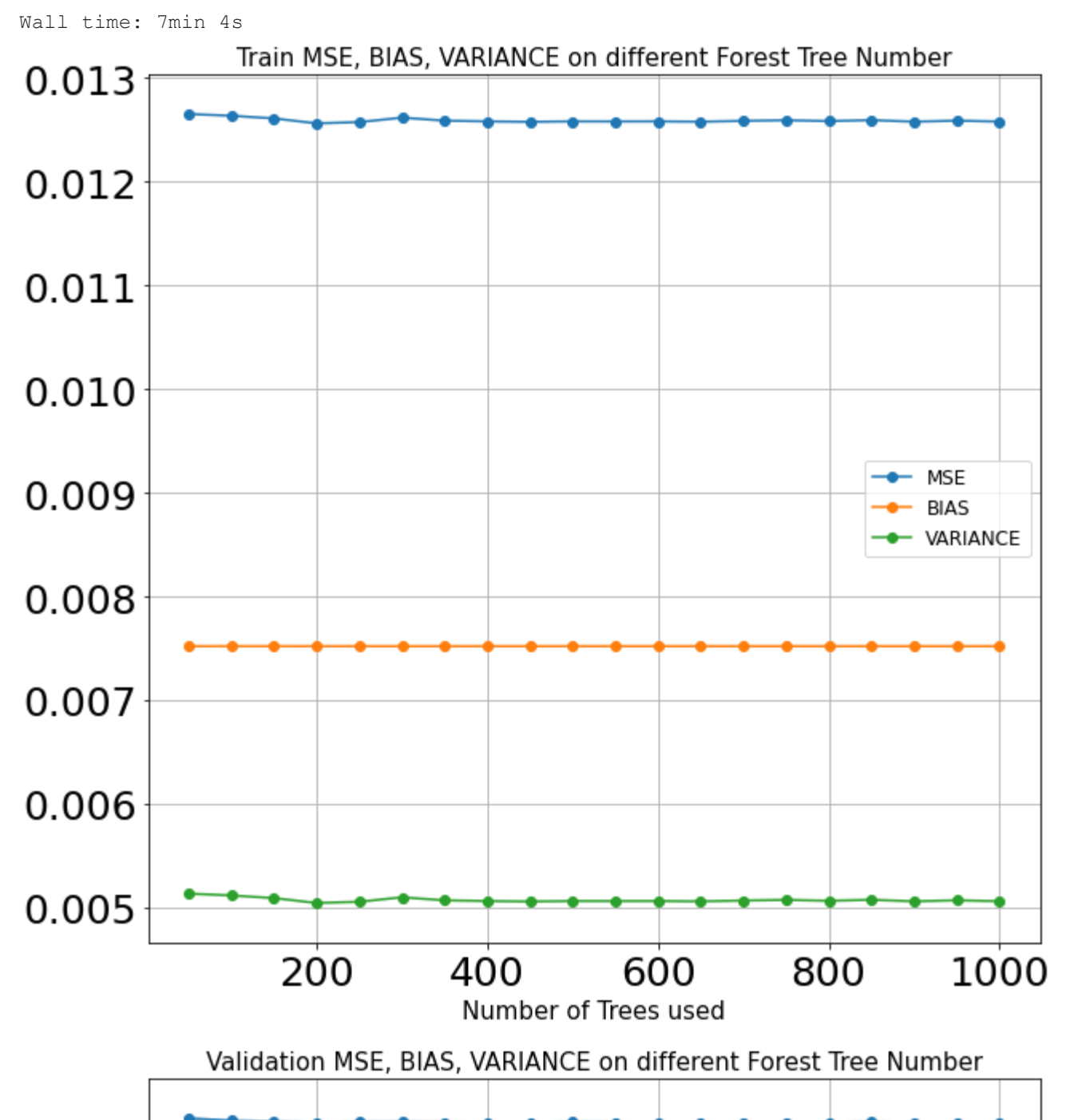
In [19]:
```
%%time
rf_model.append(
    get_rf(
        0,
        verbose = False,
        debug   = False#,
        #file_name = 'Prova1'
    )
)
```
```
Train: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA

Punteggio finale: 0.011273320418721063 (1000) alberi
Best mse: 0.011270233531289177
Best number of Trees: 750

Punteggio finale: 0.006751486653928585 (1000) alberi
Best bias: 0.006751477646661825
Best number of Trees: 200

Punteggio finale: 0.004521833764792367 (1000) alberi
Best var: 0.004518733332566147
Best number of Trees: 750

Validation: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA

Punteggio finale: 0.03028440003450135 (1000) alberi
Best mse: 0.03028271773920329
Best number of Trees: 850

Punteggio finale: 0.029518041993858414 (1000) alberi
Best bias: 0.029517623040668423
Best number of Trees: 100

Punteggio finale: 0.0007663580406430065 (1000) alberi
Best var: 0.0007647761846840253
Best number of Trees: 850

Wall time: 1h 20min 22s
```

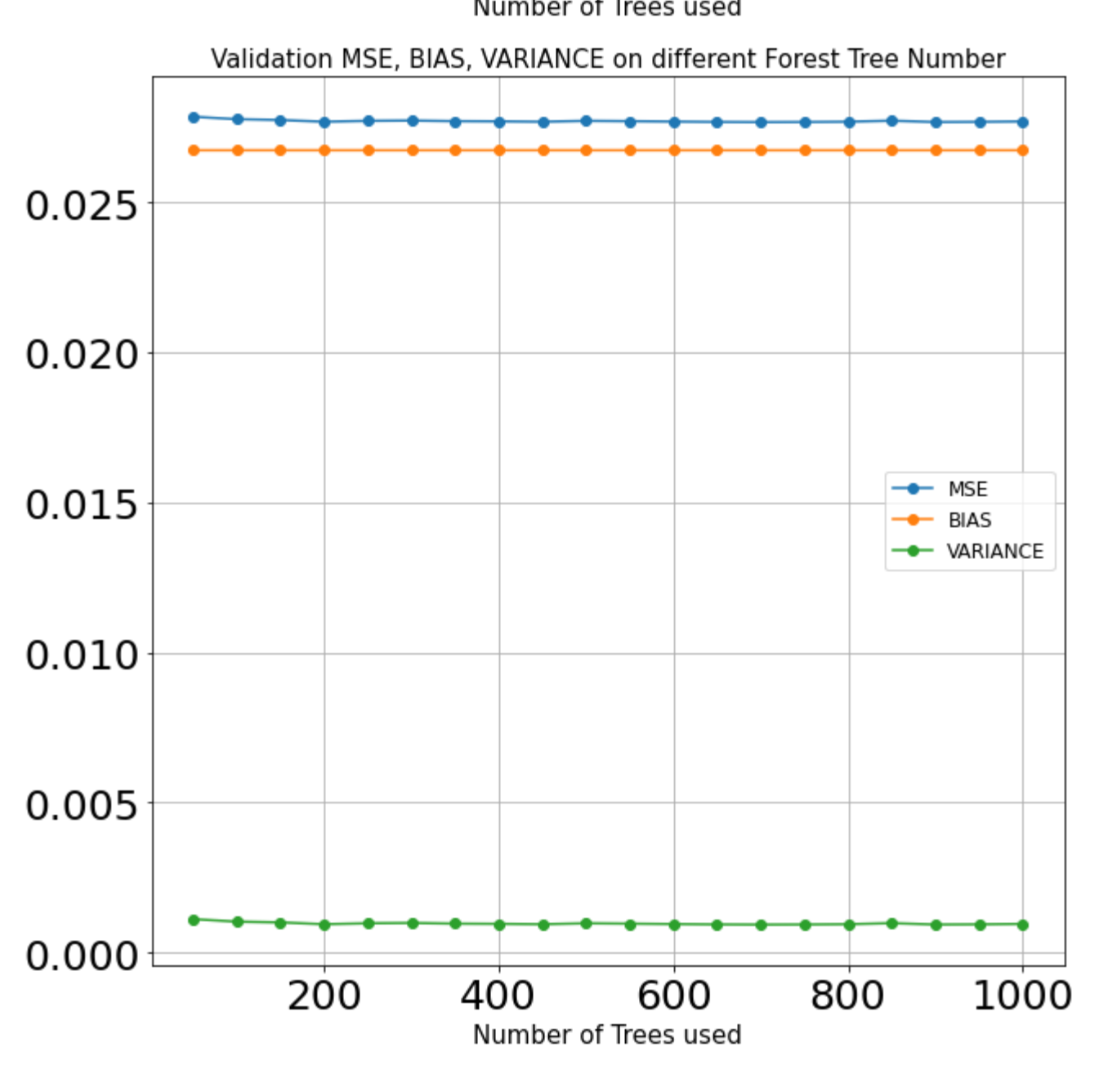




Seconda Contea 2061

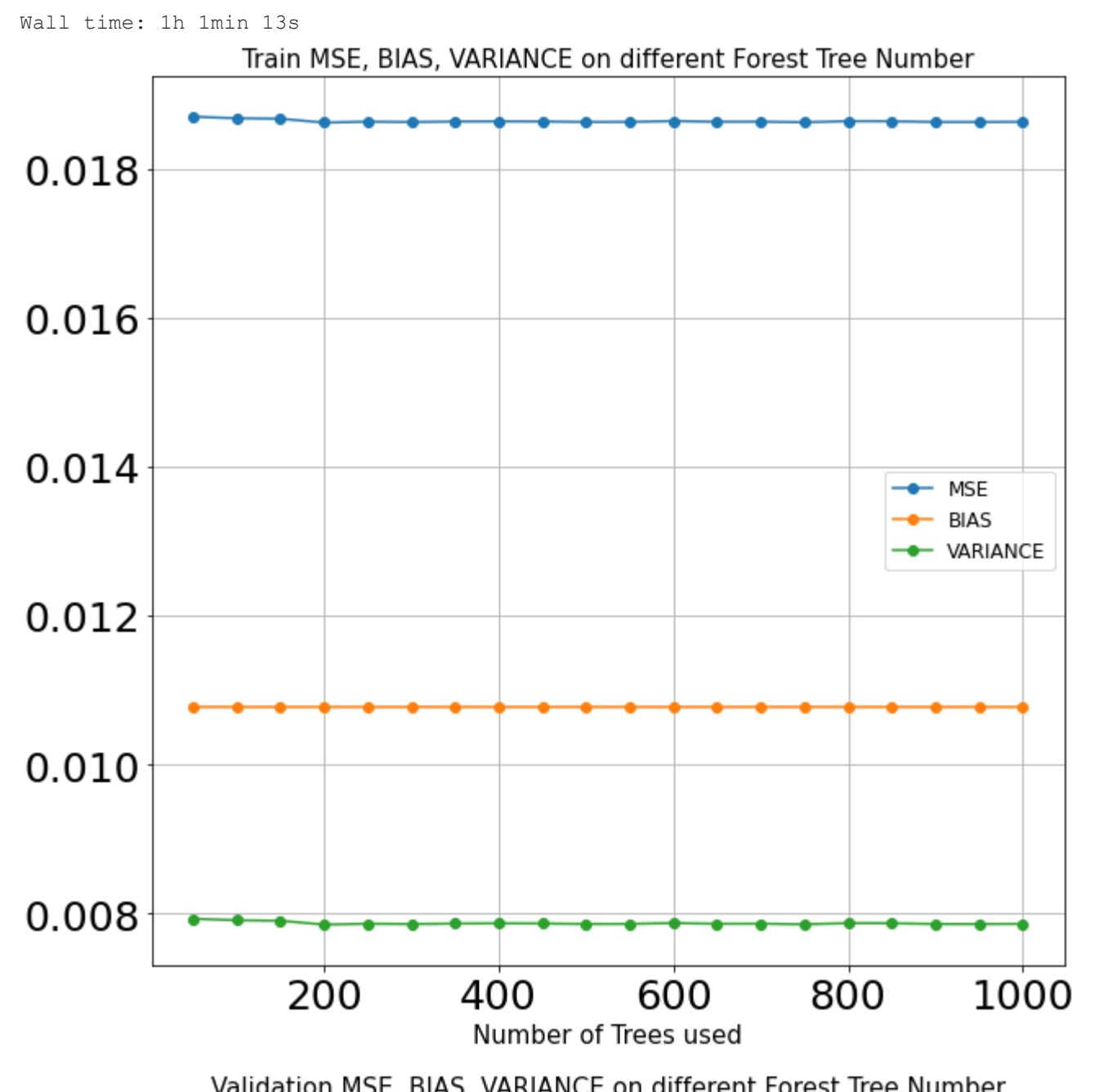
In [20]:
```
%%time
rf_model.append(
    get_rf(
        1,
        verbose = False,
        debug   = False#,
        #file_name = 'Prova2'
    )
)
```
```
Train: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA

Punteggio finale: 0.012578312287415128 (1000) alberi
Best mse: 0.012561191124724912
Best number of Trees: 200

Punteggio finale: 0.007518784142778679 (1000) alberi
Best bias: 0.007518749134962226
Best number of Trees: 50

Punteggio finale: 0.005059528144636468 (1000) alberi
Best var: 0.005042441969792701
Best number of Trees: 200

Validation: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA

Punteggio finale: 0.027682165249020445 (1000) alberi
Best mse: 0.02765956148642825
Best number of Trees: 700

Punteggio finale: 0.026722373013392222 (1000) alberi
Best bias: 0.026722210819404763
Best number of Trees: 50

Punteggio finale: 0.000959792235628269 (1000) alberi
Best var: 0.0009371592565283578
Best number of Trees: 700

Wall time: 7min 4s
```

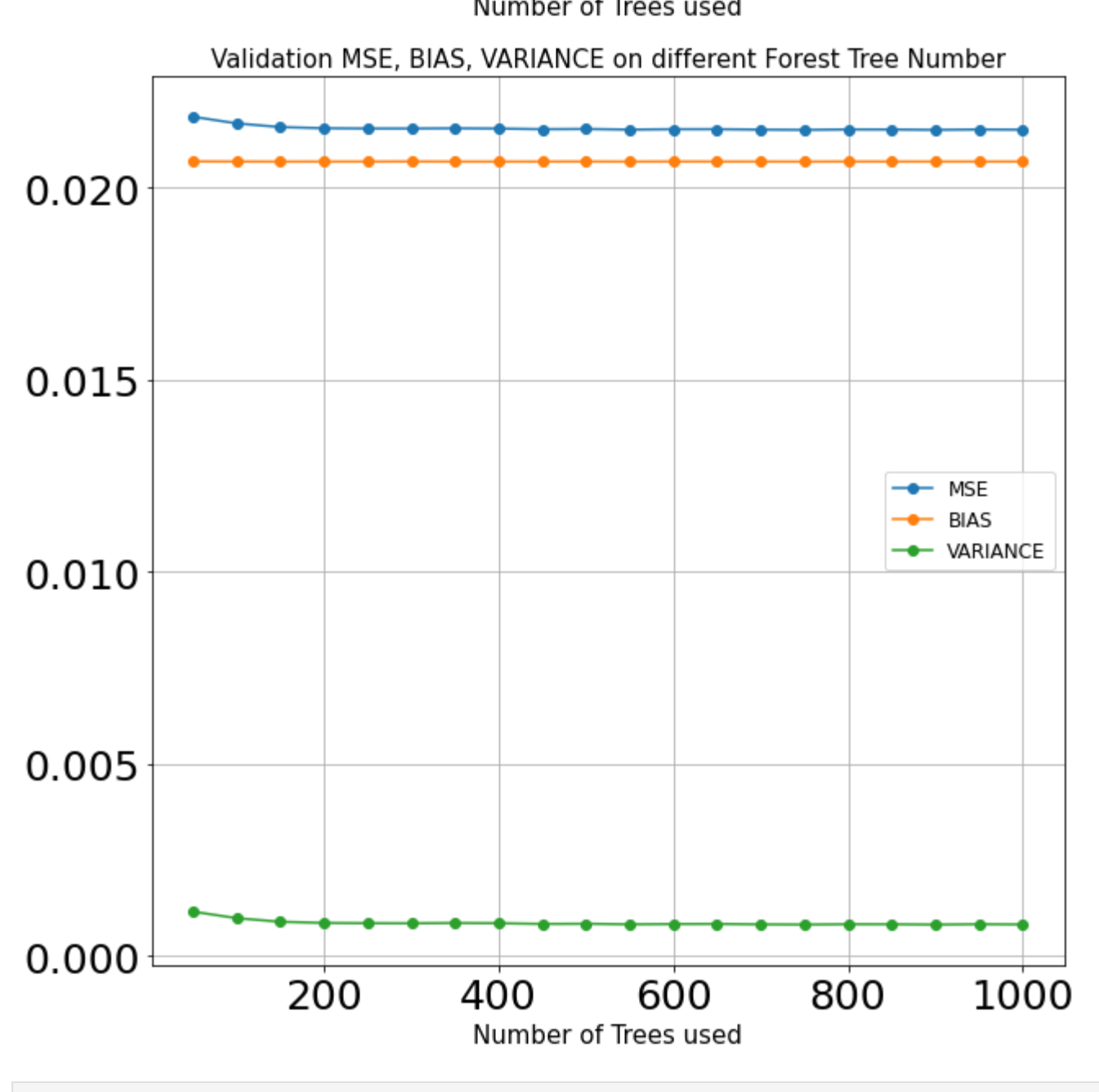




Seconda Contea 2061

In [21]:
```
%%time
rf_model.append(
    get_rf(
        2,
        verbose = False,
        debug   = False#,
        #file_name = 'Prova3'
    )
)
```
```
Train: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA

Punteggio finale: 0.018628812624583239 (1000) alberi
Best mse: 0.018618922110158235
Best number of Trees: 200

Punteggio finale: 0.010770618773541698 (1000) alberi
Best bias: 0.010770557924095961
Best number of Trees: 50

Punteggio finale: 0.007858193722906867 (1000) alberi
Best var: 0.007848279741787144
Best number of Trees: 200

Validation: TUNING DEL MASSIMO NUMERO DI ALBERI NELLA FORESTA

Punteggio finale: 0.021516305853652453 (1000) alberi
Best mse: 0.021511666932380363
Best number of Trees: 750

Punteggio finale: 0.020686515473398165 (1000) alberi
Best bias: 0.020684871155234876
Best number of Trees: 750

Punteggio finale: 0.000829790389264212 (1000) alberi
Best var: 0.0008258387941221248
Best number of Trees: 900

Wall time: 1h 1min 13s
```





In [22]:
```
rf_model
```
Out[22]:
```
[RandomForestRegressor(n_estimators=850, n_jobs=-1),
 RandomForestRegressor(n_estimators=700, n_jobs=-1),
 RandomForestRegressor(n_estimators=750, n_jobs=-1)]
```

Analisi dei risultati

I grafici delle tre contee presentano lo stesso andamento: sia nel *Train* che nel *Validation* l'errore e la sua decomposizione in $bias^2$ e *variance* sembra rimanere constante con leggerissime oscillazioni. Il bias rimane costante (soglia della predizione banale) mentre la varianza e dunque anche l'errore quadratico medio tendono a decrescere lentamente seppur non in modo significativo. Se l'andamento dovesse rimanere quello evidenziato avrebbe senso provare aumentare il numero di alberi per verificare se l'errore continua comunque a descrescere seppur non in maniera netta.

Statistiche per le tre contee su Train, Validation e Test

In [23]:
```
def print_stats(X, y, models):
    for i in range(3):
        print(f'{region_names[i]}: {get_bias_var_mse(X[i], y[i].values.ravel(), models[i])}')
        print()
```

In [24]:
```
def print_all_stats(models):
    for X, y, name in zip(
        [X_train,        X_val, X_test],
        [y_train,        y_val, y_test],
        ['Train', 'Validation', 'Test']
    ):
        print(name)
        print()
        print_stats(X, y, models)
        print()
```

In [25]:
```
print_all_stats(rf_model)
```
```
Train

A: {'bias': 0.006929732943730186, 'var': 0.0028278623503827423, 'mse': 0.009757595294112896}

B: {'bias': 0.007538416011279123, 'var': 0.0032002626832870144, 'mse': 0.010738678694566144}

C: {'bias': 0.011106016247280014, 'var': 0.005308688929980173, 'mse': 0.016414705177260284}


Validation

A: {'bias': 0.027264824537540325, 'var': 0.0007294856161013914, 'mse': 0.027994310153641653}

B: {'bias': 0.01959553796901554, 'var': 0.0009463747722312617, 'mse': 0.020541912741246834}

C: {'bias': 0.022164960473785994, 'var': 0.0008393265986282246, 'mse': 0.02300428707241416}


Test

A: {'bias': 0.028625443424184153, 'var': 0.0006180639354800561, 'mse': 0.02924350735966414}

B: {'bias': 0.024092737323504886, 'var': 0.0009170424882830598, 'mse': 0.02500977981178793}

C: {'bias': 0.033486378306030073, 'var': 0.0008326610732036911, 'mse': 0.03431903937923365}
```

Ho cottenuto un errore minore rispetto al modello del notebook precendente, ma che rimane comunque troppo alto e che non molto distante da quello del base rank.

Importanza delle feature

Analizzo il ranking delle feature delle tre foreste

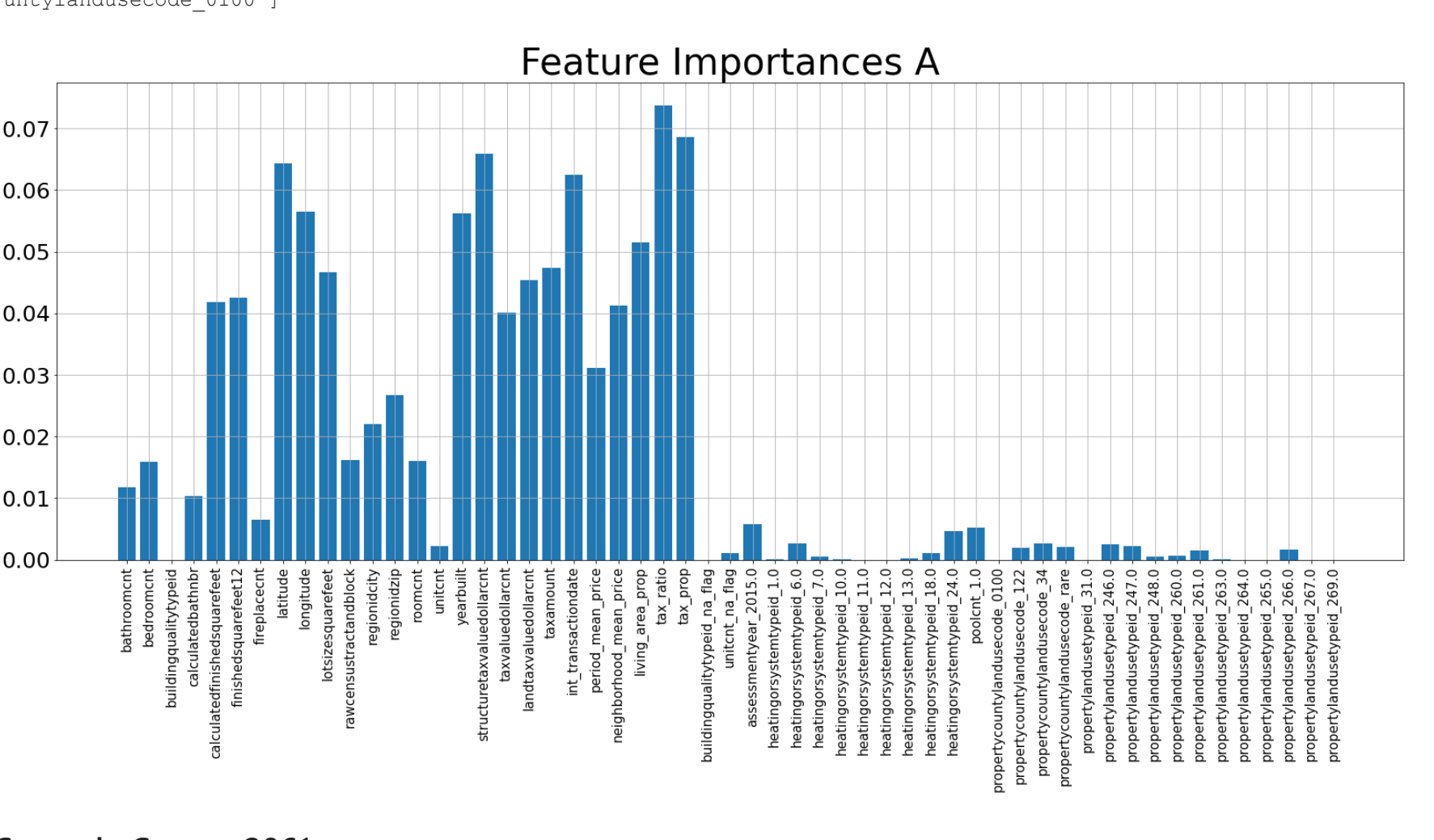
In [26]:
```
def feature_importance(X, rf, reg_name, file_name=''):

    print(f'{reg_name} FEATURE IMPORTANCES')
    print(list(X.columns[np.argsort(rf.feature_importances_)[::-1]]))
    print()

    fig, ax = plt.subplots(figsize=(len(rf.feature_importances_)/2,10))

    ax.tick_params(axis='x', which='major', labelsize=15)
    ax.tick_params(axis='x', which='minor', labelsize=20)
    ax.tick_params(axis='y', which='major', labelsize=25)
    ax.tick_params(axis='y', which='minor', labelsize=30)

    ax.bar(range(0, X.shape[1]), rf.feature_importances_)
    ax.set_title(f'Feature Importances {reg_name}')
    ax.set_xticks(range(X.shape[1]))
    ax.set_xticklabels(X.columns, rotation=90)

    ax.grid()

    if file_name != '':
        fig.savefig('images/' + file_name + '_RandomForestRegressor_FeatureImportance.jpg')
```

In [27]:
```
def get_feature_importance(index, file_name=''):
    if file_name == '':
        file_name = region_ids[index]
    feature_importance(
        X_train[index],
        rf_model[index],
        region_names[index],
        file_name = file_name
    )
```

Prima Contea 1286

In [28]:
```
get_feature_importance(0)
```

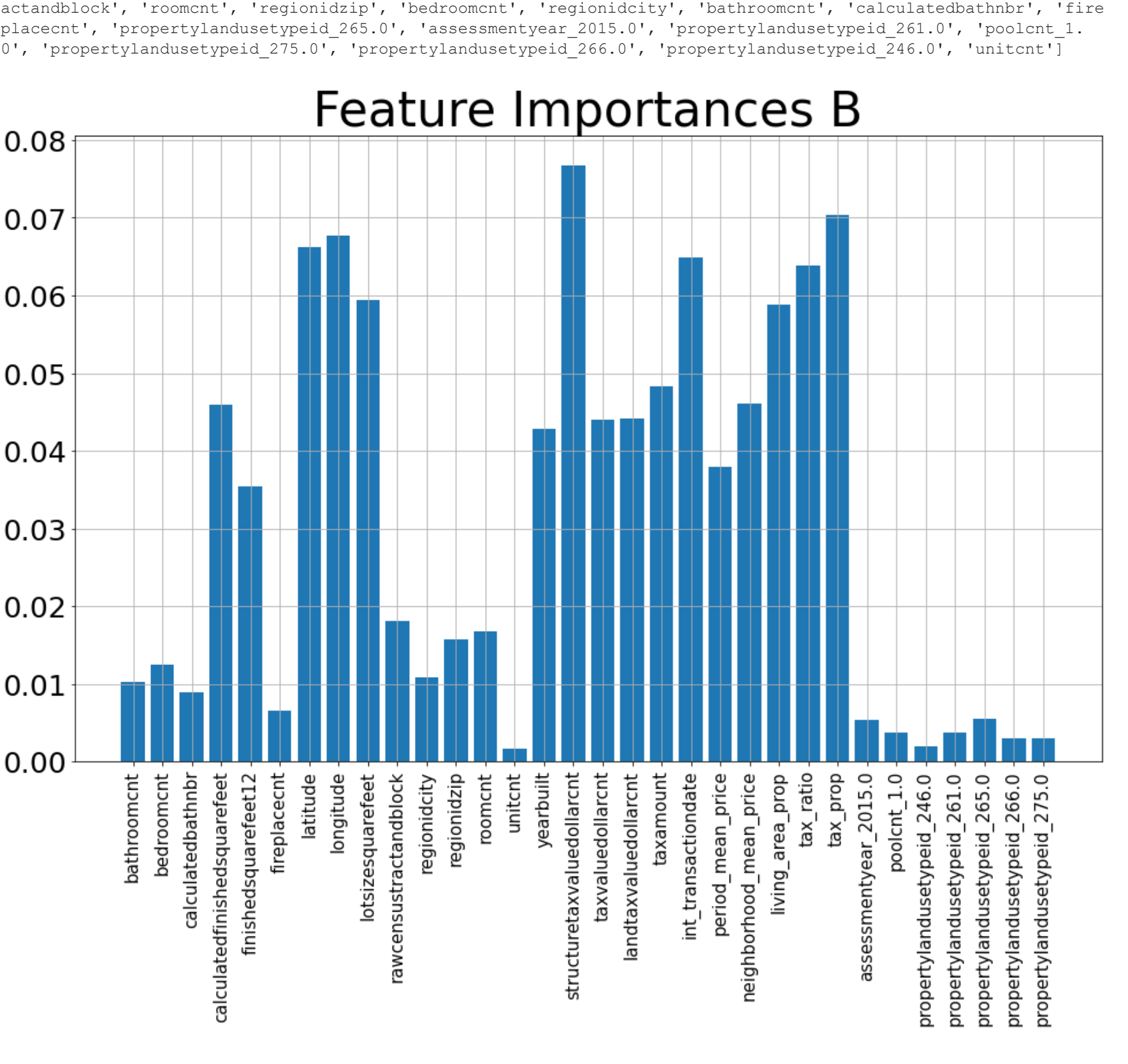
```
A FEATURE IMPORTANCES
['tax_ratio', 'tax_prop', 'structuretaxvaluedollarcnt', 'latitude', 'int_transactiondate', 'longitude', 'yearbuilt', 'living_area_prop', 'taxamount', 'lotsizesquarefeet', 'landtaxvaluedollarcnt', 'finishedsquarefeet12', 'calculatedfinishedsquarefeet', 'neighborhood_mean_price', 'taxvaluedollarcnt', 'period_mean_price', 'regionidzip', 'regionidcity', 'rawcensustractandblock', 'roomcnt', 'bedroomcnt', 'bathroomcnt', 'calculatedbathnbr', 'fireplacecnt', 'assessmentyear_2015.0', 'poolcnt_1.0', 'heatingorsystemtypeid_24.0', 'propertycountylandusecode_34', 'heatingorsystemtypeid_6.0', 'propertylandusetypeid_246.0', 'propertylandusetypeid_247.0', 'unitcnt', 'propertycountylandusecode_rare', 'propertycountylandusecode_122', 'propertylandusetypeid_266.0', 'propertylandusetypeid_261.0', 'unitcnt_na_flag', 'heatingorsystemtypeid_18.0', 'propertylandusetypeid_260.0', 'propertylandusetypeid_248.0', 'heatingorsystemtypeid_7.0', 'heatingorsystemtypeid_13.0', 'heatingorsystemtypeid_1.0', 'propertylandusetypeid_263.0', 'heatingorsystemtypeid_10.0', 'heatingorsystemtypeid_11.0', 'propertylandusetypeid_269.0', 'heatingorsystemtypeid_12.0', 'propertylandusetypeid_264.0', 'propertylandusetypeid_265.0', 'buildingqualitytypeid', 'buildingqualitytypeid_na_flag', 'propertylandusetypeid_31.0', 'propertylandusetypeid_267.0', 'propertycountylandusecode_0100']
```



Seconda Contea 2061

In [29]:
```python
get_feature_importance(1)
```

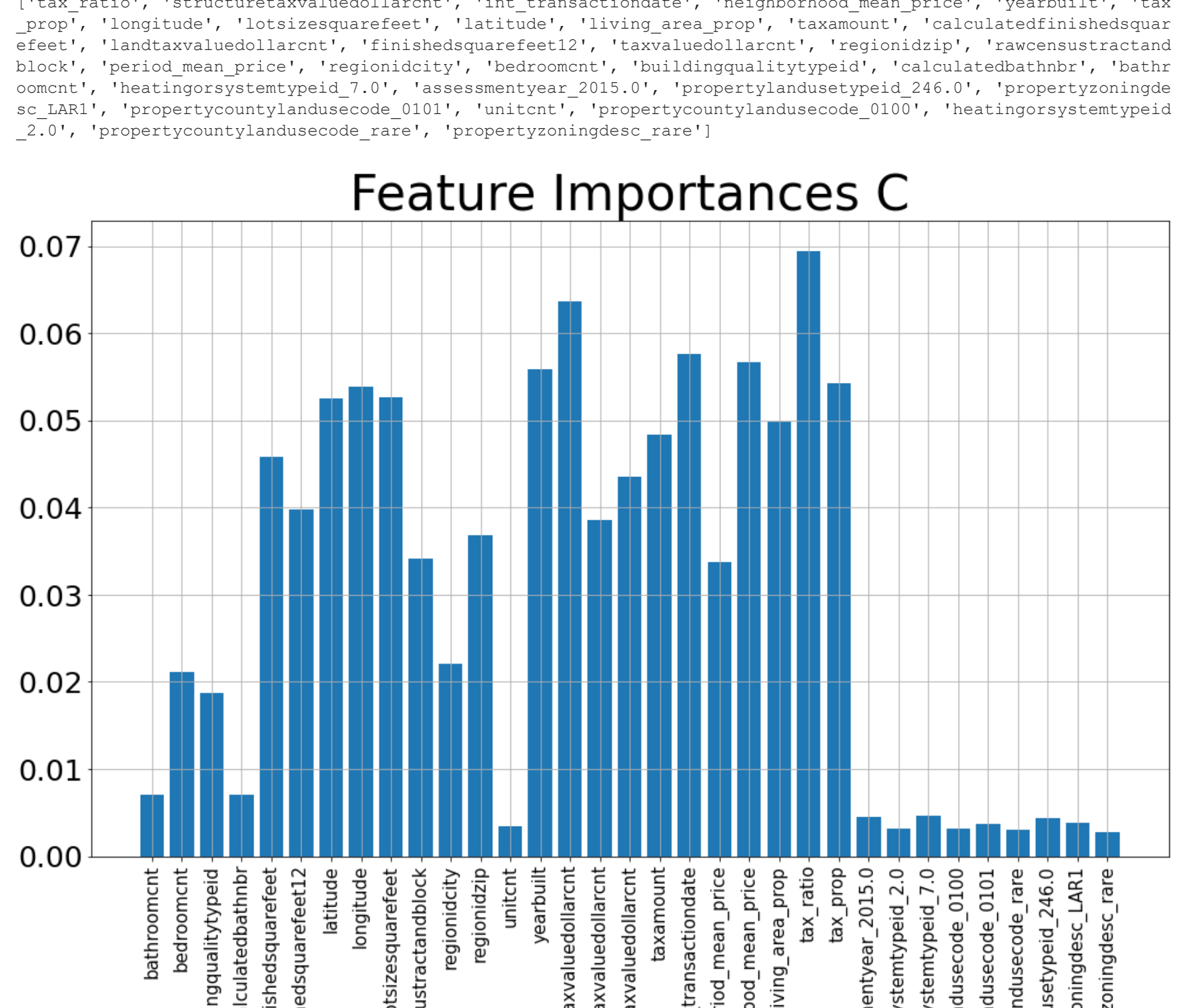
```
B FEATURE IMPORTANCES
['structuretaxvaluedollarcnt', 'tax_prop', 'longitude', 'latitude', 'int_transactiondate', 'tax_ratio', 'lotsizesquarefeet', 'living_area_prop', 'taxamount', 'neighborhood_mean_price', 'calculatedfinishedsquarefeet', 'landtaxvaluedollarcnt', 'taxvaluedollarcnt', 'yearbuilt', 'period_mean_price', 'finishedsquarefeet12', 'rawcensustractandblock', 'roomcnt', 'regionidzip', 'bedroomcnt', 'regionidcity', 'bathroomcnt', 'calculatedbathnbr', 'fireplacecnt', 'propertylandusetypeid_265.0', 'assessmentyear_2015.0', 'propertylandusetypeid_261.0', 'poolcnt_1.0', 'propertylandusetypeid_275.0', 'propertylandusetypeid_266.0', 'propertylandusetypeid_246.0', 'unitcnt']
```



Terza Contea 3101

In [30]:
```python
get_feature_importance(2)
```

```
C FEATURE IMPORTANCES
['tax_ratio', 'structuretaxvaluedollarcnt', 'int_transactiondate', 'neighborhood_mean_price', 'yearbuilt', 'tax_prop', 'longitude', 'lotsizesquarefeet', 'latitude', 'living_area_prop', 'taxamount', 'calculatedfinishedsquarefeet', 'landtaxvaluedollarcnt', 'finishedsquarefeet12', 'taxvaluedollarcnt', 'regionidzip', 'rawcensustractandblock', 'period_mean_price', 'regionidcity', 'bedroomcnt', 'buildingqualitytypeid', 'calculatedbathnbr', 'bathroomcnt', 'heatingorsystemtypeid_7.0', 'assessmentyear_2015.0', 'propertylandusetypeid_246.0', 'propertyzoningdesc_LAR1', 'propertycountylandusecode_0101', 'unitcnt', 'propertycountylandusecode_0100', 'heatingorsystemtypeid_2.0', 'propertycountylandusecode_rare', 'propertyzoningdesc_rare']
```



La distribuzione d'importanza è ben distribuita, seppur spicchino comumque le feature inerenti alle tasse, capaci probabilmente di considerare informazioni di natura differente. Le feature aggiunte in fase di preparazione hanno un'importanza rilevante.